TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Nástroj pro vizualizace časových řad

Bakalářská práce

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Michal Dvořák**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.
Ústav nových technologií a aplikované informatiky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Nástroj pro vizualizace časových řad

Jméno a příjmení: **Michal Dvořák**
Osobní číslo: M18000072
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Seznamte se s problematikou vizualizace časových řad a s tvorbou interaktivních webových uživatelských rozhraní.
2. Navrhněte webovou aplikaci pro vizualizaci časových řad s možností aplikace vybraných filtrů a dalších funkcí. Při návrhu se zaměřte na snadnou rozšiřitelnost funkcí a rychlost zpracování větších datových sad.
3. Návrh prakticky implementujte, kód přehledně zdokumentujte a pokryjte testy tak, aby byl dobrý předpoklad pro jeho další využití a rozšiřitelnost.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah pracovní zprávy: 30 – 40 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] MDN JavaScript [online]. Mozilla foundation [cit. 2020-10-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
[2] DALE, Kyran. Data visualization with Python and JavaScript. Beijing: O'Reilly, 2016. ISBN 978-1491920510.
[3] OSMANI, Addy. Learning JavaScript design patterns. Sebastopol, CA: O'Reilly Media, 2012. ISBN 978-1449331818.

Vedoucí práce: Mgr. Jiří Vraný, Ph.D.
Ústav nových technologií a aplikované informatiky

Datum zadání práce: 19. října 2020
Předpokládaný termín odevzdání: 17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2020

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

16. května 2021 Michal Dvořák

Nástroj pro vizualizace časových řad

Abstrakt

Cílem této práce je tvorba nástroje pro vizualizaci časových řad, respektive zpracování csv souborů obsahující data, která se měnila v čase, a přehledně je zobrazovat. Jako řešení byla vytvořena webová aplikace, postavená na Javascriptu, využívající hlavně knihovnu D3.js. Pro účely práce byla vytvořena i serverová backend aplikace, využívající prostředí Node.js, k usnadnění manipulace s daty a možnosti uchovávat dlouhodobě data v databázi. Celá aplikace tak poskytuje jednoduché, ale efektivní nástroje pro manipulaci s daty, jejich zobrazování do grafů, a také nástroje pro práci s grafy samotnými, které v konkurenčních řešení mnohdy chybí.

Klíčová slova: vizualizace, časových řad, data, v čase, webová aplikace, D3.js, Node.js, csv, server

Abstract

Goal of this project is to create a tool that visualizes time series data, or more accurately a tool that processes csv files containing data that has changed over time, and show them clearly to a user. Whole web application was made as a solution. This web application is built on JavaScript and mainly uses D3.js library. For needs of this project backend server application was made, so data manipulation is easier and so whole project can store data in long term database. Whole application then provides easy yet effective tools to manipulate data, display them in plots and it also provides tools that work with plots themself, that are mostly missing in competition's solutions.

Keywords: visualization, time series, data, web, application, D3.js, Node.js, csv, server

Poděkování

Chtěl bych poděkovat svému vedoucímu práce, panu Mgr. Jiřímu Vranému, Ph.D., za možnost vypracovávat tuto bakalářskou práci, kterou, doufám, on a jeho tým ocení a využijí při svém výzkumu. Zároveň bych mu chtěl poděkovat za jeho trpělivost, ochotu a věcné rady během našich konzultací.

Obsah

Seznam zkratek

CSS	Cascading Style Sheets, jazyk popisující, jak má webová stránka vypadat
CSV	Comma-separeted value, značí formátování dat, kdy jednotlivé hodnoty jsou od sebe oddělené čárkou
DOM	Data object model, stromová struktura jednotlivých částí nejen HTML dokumentů
HTML	Hyper-text markup language, popisovací jazyk, kterým se popisuje struktura webových stránek
HTTP	Hypertext Transfer Protocol, protokol popisující přenos dat internetem
JS	JavaScript, skriptovací programovací jazyk
JSON	JavaScript Object Notation, formát pro výměnu a uchovávání dat
MVC	Model-View-Controller, architektura webových aplikací
PNG	Portable Network Graphics, formát rastrové grafiky
REST	Representational state transfer, architektura webových aplikací
URI	Uniform Resource Identifier, identifikátor přesně definující zdroj informací

1 Úvod

Tato bakalářská práce se zabývá tvorbou webové aplikace pro vizualizaci časových řad. Pod pojmem „časová řada“ si lze například představit v kontextu této práce uložené hodnoty z měření rychlosti auta. Taková časová řada tedy obsahuje údaje o aktuální rychlosti automobilu v časech od nastartování motoru, až po jeho vypnutí. Výsledná webová aplikace si klade za cíl data tohoto charakteru zobrazovat přehledně do grafů a umožňovat uživateli manipulaci s nimi tak, aby byl schopen zjistit aktuální hodnotu v kterémkoliv úseku grafu. Výsledná webová aplikace se bude využívat při výzkumu, který probíhá na univerzitě a je součástí projektu Doprava 2020+ vypsaným Technologickou agenturou České republiky[1]. Webová aplikace má ale za cíl, být použitelná i pro jiná různá data, která nepocházejí z tohoto výzkumu, a aplikace by tak mohla být přínosná i pro jiné projekty.

Důvod vzniku takovéto aplikace, je jednoduchý. Neexistuje žádná dohledatelná webová aplikace, která by odpovídala nárokům týmu pracujícím na projektu Doprava 2020+. Tým nepotřebuje jen data zobrazit, ale potřebuje mít i možnost zobrazit pouze úsek vykreslených dat, který lze poté jednoduše uložit, tak aby se k tomuto konkrétnímu úseku šlo jednoduše vrátit. Dalším častým nedostatkem konkurenčních řešení je špatná optimalizace, či úplná nemožnost práce s tak rozsáhlými časovými řadami, jako právě mohou být data sbírané za chodu automobilu. Taková data mají rozsah až kolem sta unikátních sledovaných hodnot s jednotkami tisíc zaznamenaných hodnot, podle délky sbírání dat. Dalším nárokem je poté možnost nad načtenými daty provádět funkce modifikující hodnoty, jako například výpočet klouzavého průměru a dalších.

V první části práce budou krátce popsány čtyři javascriptové knihovny pro vizualizaci dat. Konkrétně knihovny AnyChart.js, D3.js, Chart.js a Plotly.js. Dále budou v práci zmíněné konkurenční aplikace, které řeší podobný problém, ale nejsou dostatečné, ať už kvůli důvodům popsaných v předešlém odstavci nebo kvůli nějakým jiným.

Dále budou v práci popsány konkrétní nároky na aplikaci, návrh vhodného řešení, které vyhovuje požadavkům a budou zde také popsány zvolené technologie pro vývoj aplikace. Následně největší část práce bude věnována procesu tvorby webové aplikace a popisu jejích částí, aby na aplikaci šlo následně navázat dalším vývojem, bude-li to třeba. V závěru poté budou shrnuty výsledky práce, diskuze o vytvořeném řešení a návrhy nad dalším vývojem.

2 Aktuální technologie používané k vizualizaci dat na webu

Aktuálně se k tvorbě webových stránek se stále využívají tři hlavní technologie: HTML, CSS a Javascript. HTML v nejnovější verzi 5 existuje již 13 let[2] a CSS verze 3 existuje od roku 2011[3]. Javascript se naproti tomu neustále vyvíjí, jelikož nové verze ECMAScriptu, z kterého jazyk vychází, se publikují každý rok. Jazyk Javascript ale jako takový je tu od roku 1995, a jeho verze vycházející z ECMAScriptu 5.1, která zpopularizovala programování v Javascriptu, je již 10 let stará[4].

Vývojáři webových aplikací si pro pokročilé funkce vytváří vlastní knihovny s použitím těchto technologií. Ty poté mezi sebou knihovny sdílí, a dokonce se spojují a společně na knihovnách pracují a vyvíjí je. Velké společnosti, které využívají tyto knihovny, poté často financují právě vývoj těchto knihoven a projektů. Na Githubu[5], kde se tyto knihovny nejčastěji nachází, existuje celkem 128 milionů veřejných projektů[6], a z toho nejvíce patří právě Javascriptu, který zde má zastoupení 19 %[7]. To celkem dělá 24 milionů veřejných Javascriptových projektů. Samozřejmě ne vše jsou jen knihovny ulehčující práci dalším programátorům, kdyby ale z těch 24 milionů bylo jen 1 % opravdu knihoven pro práci s Javascriptem, tak existuje 240 tisíc knihoven dostupných pro Javascript.

Různé knihovny ulehčují práci s tvorbou webových stránek v různých směrech. Tato práce se ale bude zabývat pouze těmi, které ulehčují tvorbu a zprostředkovávají funkcionality pro vizualizaci dat. I v dnešní době je ale možné vizualizovat data jen za použití čistého Javascriptu, HTML a CSS, nicméně v době, kdy jsou programátorům dostupné tolik velký ověřených knihoven, je zbytečné znovu vynalézat kolo.

Následující podkapitoly se budou týkat srovnáním čtyř knihoven pro vizualizaci dat, které se často objevují v článcích jako jedny z uživatelsky nejoblíbenějších. Čerpal jsem ze serverů DZone[8] a Sitepoint[9]. U těchto knihoven je důležité jak se používají z pohledu programátora, jaké poskytují možnosti vizualizace a jak se na jejich základě dá postavit robustní webová aplikace.

2.1 Knihovna AnyChart.js

Javascriptová knihovna AnyChart.js je robustní knihovna, která cílí na jednoduchou tvorbu různých typů grafů a vizualizaci dat. Podle oficiálních stránek knihovny začal její vývoj už v roce 2003 a za tu dobu získala uznání od velkých společností jako je Oracle, Microsoft či Samsung [10].

Knihovna má rozsáhlé API, které je velice dobře zdokumentované a na samotných stránkách lze dohledat i živé ukázky kódu a návody, jak knihovnu využívat. Samotné použití je knihovny je opravdu jednoduché, například graf na Obrázku 2.1 lze vytvořit pouze 30 řádky kódu.

Obrázek 2.1: Ukázka grafu vytvořeného knihovnou AnyChart.js

Knihovna umožňuje tvořit i různé jiné typy grafů, jako je například koláčový graf, sloupcový graf, bublinový graf a spoustu dalších. Umožňuje dokonce tvorbu map, kam lze vkládat body a hodnoty pro vizualizaci například počtu obyvatel ve velkých městech. Knihovna zvládá i vykreslovat velké množství dat, které se vyvíjí v čase, na svých stránkách to vývojáři prezentují na vývoji cen akcií na burze. AnyChart.js také poskytuje desítky předpřipravených vzhledů a možností, jak data vykreslit. Pokud chce ale uživatel něco změnit podle svého, co vybočuje z filozofie a designu knihovny, dostupné API už není moc nápomocné.

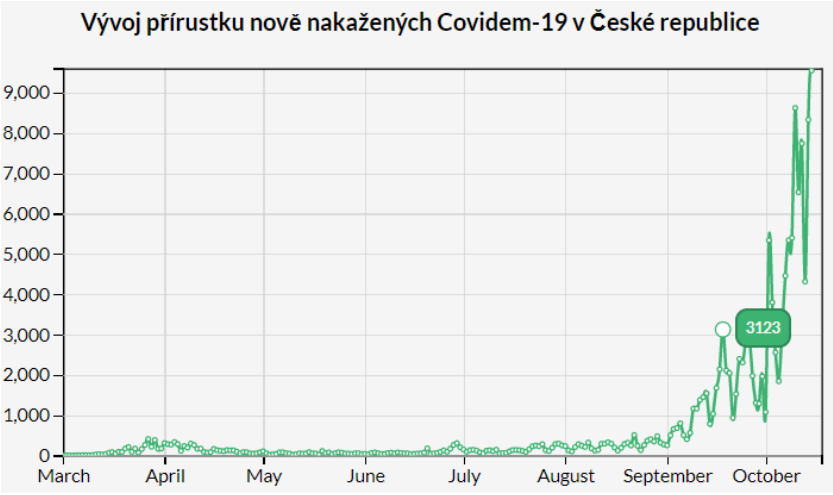
Nevýhodou AnyCharts může být jejich finanční model. Knihovna je totiž zdarma pouze pro nekomerční a studijní využití. Tato verze zdarma je označována jako Trial verze, což je vidět na Obrázku 2.1 v levém dolním rohu. AnyCharts totiž v trial verzi vkládá do grafů informaci o tom, že se jedná pouze o trial verzi. Funkčně se ale trial verze od placené verze nijak neliší.

2.2 Knihovna D3.js

Název D3 zkracuje název „Data-driven documents“. Knihovna jako taková totiž primárně necílí na vizualizaci dat, ale na manipulaci webových dokumentů v návaznosti na propojená data. Knihovna totiž umožňuje provázat DOM webové stránky s libovolnými daty, a podle těchto dat generovat obsah nebo ho měnit se změnou dat[11] V kontextu vizualizace dat to znamená, že knihovna může provázat SVG element se vstupními daty a ty poté vykreslit do grafu podle přání programátora. Poté pokud se z řady vstupních dat odeberou nějaké hodnoty ze začátku a z konce, vznikne jakýsi výřez a knihovna se postará o přizpůsobení grafu, tak aby obsahoval jen hodnoty z toho výřezu. Tato funkce ale nemusí být vůbec použita jen na zobrazování dat, ale lze takto generovat celou stránku, například nadpisy, tlačítka a tak dále.

Knihovna se skládá z různých modulů a řada z nich je cílená právě na samotnou vizualizaci dat. Jeden modul se například stará o dynamickou tvorbu os grafu, další poté o vykreslení hodnot do grafu, další zas o proložení dat křivkou. Moduly knihovny D3.js jsou velice obsáhlé a kolem knihovny tak vznikla velká komunita uživatelů.

D3.js poskytuje ke svému kódu velice rozsáhlou dokumentaci, kde jsou popsány všechny moduly a jejich funkce, takže ačkoliv práce s D3.js není moc intuitivní, programátor má možnost si všechny potřebné informace jednoduše dohledat. Součástí dokumentace jsou i návody, jak na základní práci s knihovnou, tak i na její pokročilé funkce. D3.js také poskytuje galerii skoro dvou set kódů, napsaných pomocí jejich knihovny, řešící široké spektrum problémů od generování jednoduchých přímkových grafů, přes vizualizaci grafových struktur po vytváření 3D map vesmíru společně se zobrazováním údajů o vesmírných tělesech.[12]



Obrázek 2.2: Ukázka grafu vytvořeného knihovnou d3.js

Pro používání knihovny je tedy potřeba se s ní nejdříve naučit pracovat, ale poté poskytuje programátorovi prostředí, které mu ulehčí tvorbu téměř čehokoliv, čeho

jsou dnešní webové technologie možné. Z hlediska náročnosti, graf na obrázku 2.2 je vytvořen za pomocí 150 řádků kódu i s HTML a CSS kódem.

2.3 Knihovna Chart.js

Vývojáři popisují tuto knihovnu na svých stránkách jako „Jednoduché ale flexibilní kreslení grafů Javascriptem pro designéry a vývojáře.“[13] Samotná knihovna je opravdu jednoduchá. Nabízí 8 různých typů grafů, které jsou jednoduše animovatelné a lze jejich vzhled jednoduše nastavit podle potřeb.

Práce s knihovnou prakticky spočívá ve vytvoření instance objektu Chart, který přijímá rozsáhlou konfiguraci v podobě dalšího Javascriptového objektu. Práce s grafem za běhu aplikace poté probíhá voláním funkcí instance Chart.

Ačkoliv celá práce s knihovnou spočívá pouze s vytvořením grafu, a poté případně volání jeho funkcí, tak samotná konfigurace grafu není příliš intuitivní. Vytvořit jednoduchý graf je otázka pár řádků kódu, ovšem problém poté nastává v nastavení dalších funkcionalit jako je zobrazení aktuální hodnoty pod kurzorem nebo volba výseku z grafu.

Limitem knihovny může být omezení pouze na 8 typů grafů, které jsou: přímkový graf, sloupcový graf, radarový graf, koláčový graf, polární graf, bublinkový graf, bodový graf a plošný graf. Dalším limitem je nemožnost úpravy grafu mimo poskytované možnosti nastavení, což může být problém u složitějších projektů, které potřebují upravovat grafy.

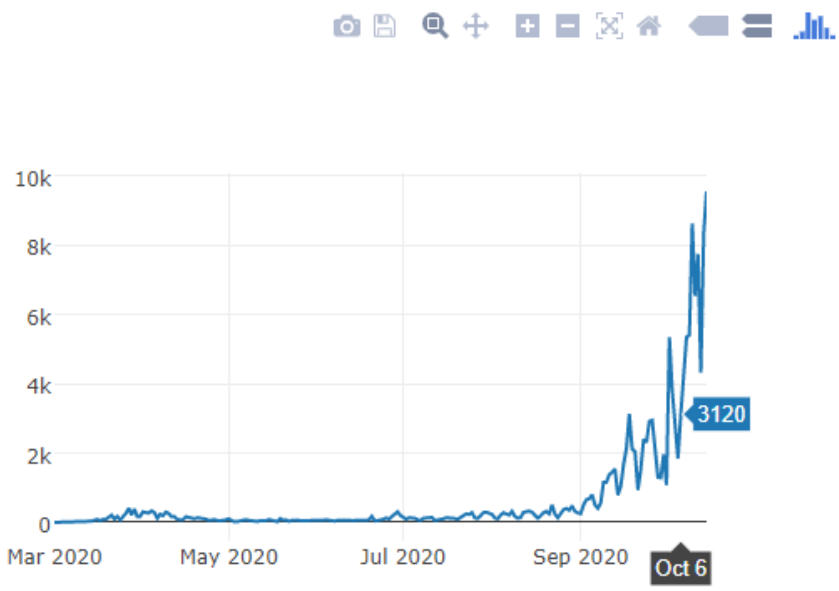
Nutno říci, že tuto knihovnu lze rozšířit použitím buď oficiálních, svých nebo cizích pluginů. Tyto pluginy přidávají další funkcionality a je tak možné dosáhnout i pokročilých funkcí pomocí této knihovny. Pro přehled složitosti, vytvořit graf na obrázku 2.3 je otázka konfigurace na 40 řádků.

Obrázek 2.3: Ukázka grafu vytvořeného knihovnou Charts.js

Knihovna Plotly.js

Knihovna Plotly.js je open-source knihovna vybudovaná nad knihovnami D3.js a stack.gl, což je knihovna pro zobrazování 3D objektů na webu.[14] Kombinace těchto dvou knihoven dovoluje vytvářet opravdu hezky vypadající grafy a jiné vizualizace.

Používání Plotly je jednoduché a práce s ní trochu připomíná práci s Charts.js, která je zmíněna v předešlé kapitole. Používání knihovny nejčastěji totiž zahrnuje vytváření Javascriptových objektů s atributy podle dokumentace a následně jejich správné skládání do sebe. Objekt grafu, tak například přijímá jako parametry instanci objektu reprezentující přímkový graf, dále instanci objektu reprezentující osy grafu, dále legendu a tak dále. Poté jen stačí objektu grafu předat element, pod kterým má být graf v DOM struktuře umístěn a knihovna graf vykreslí podle konfigurace. Například graf na obrázku 2.4 je vytvořen jen 7 řádky Javascriptového kódu z připravených dat.



Obrázek 2.4: Ukázka grafu vytvořeného knihovnou Plotly.js

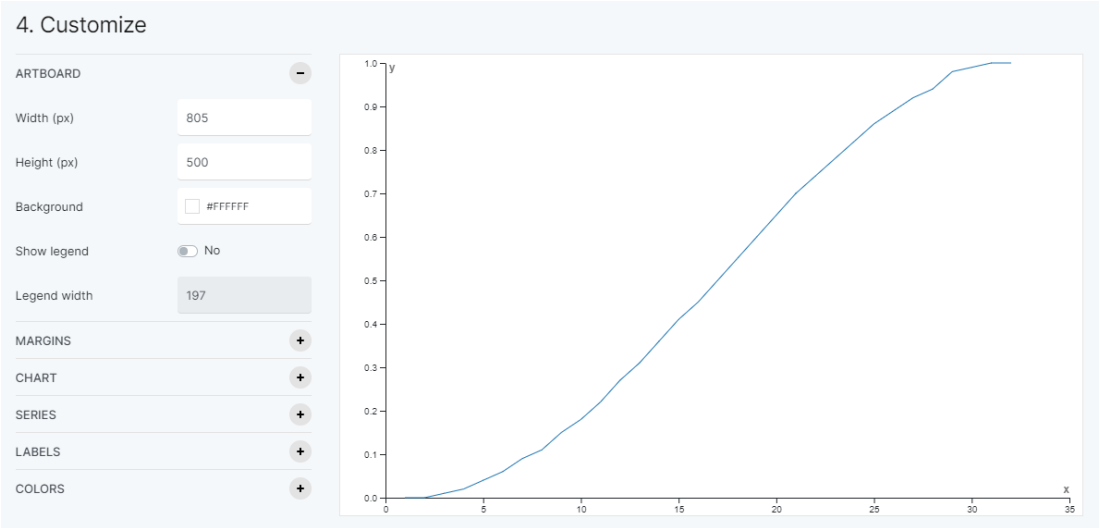
V pravém horním rohu obrázku 2.4 si lze všimnout nástrojového panelu, který Plotly v základním nastavení přidává do grafů. Graf lze tak bez žádné konfigurace a programování ukládat do obrázku, vybírat si výseky grafu, přibližovat a oddalovat graf, a i měnit nastavení zobrazení hodnot. Plotly tak poskytuje možnosti, jak velice jednoduše a elegantně vizualizovat široké spektrum dat a poskytovat k nim i spoustu užitečných funkcí.

Nevýhodou ale může být práce s grafy mimo poskytované API. Knihovna sice poskytuje rozhraní vlastních funkcí a událostí, na které může programátor v kódu reagovat, ale pokud je jeho záměrem něco změnit mimo poskytovanou konfiguraci, tak nejspíš narazí na komplikace.

3 Konkurenční řešení

3.1 RAWGraphs

RawGraphs je webová aplikace, která je zadarmo k použití a umožňuje vizualizovat data, která uživatel poskytne buď ve formátu csv anebo ve formátu JSON.[15] Aplikace má jednoduché uživatelské prostředí, které uživatele intuitivně provede celým nastavovacím procesem od nahrání vstupních dat, přes výběr hodnot k vizualizaci, až po výběr typu grafu a zobrazení výsledku.



Obrázek 3.1: Ukázka prostředí RAWGraphs.io

Na obrázku číslo 3.1 je vidět, jaký generuje aplikace výstup pro csv soubor obsahující jednoduchou časovou řadu. Výstup je velice jednoduchý přímkový graf s osami umožňující číst hodnoty grafu. Vlevo od grafu lze vidět jednoduché uživatelské prostředí umožňující nastavení výsledného vzhledu grafu. Podobně jednoduché a přehledné prostředí je i u předchozích kroků nastavovacího procesu.

Problémem RawGraphs je ale to, že to je právě jedna z aplikací umožňující pouze tvorbu statického obsahu. Výsledný graf nezobrazuje aktuální hodnotu časové řady pod polohou kurzoru. Nedovoluje ani zobrazení pouze některé části grafu. Jednoduše vykreslí graf a ten si může uživatel stáhnout a dále někde použít.

Aplikace ani neumožňuje vložená data nijak upravovat, například nějakou funkcí. Dokonce ani neumožňuje zobrazit neanotovaná data. Pokud aplikaci nejsou poskytnuty hodnoty x a y, tak graf prostě nezobrazí, což může být problém, když máme

pouze hodnoty y, které se vyvíjí v čase pořád se stejným časovým krokem, který ale není k dispozici mezi daty.

Aplikace je určitě inspirativní svým jednoduchým a přímočarým designem, ale bohužel neposkytuje žádané funkce a je proto absolutně nevhodná pro účely této práce, protože zkrátka jen dokáže data zobrazit.

3.2 Plotly - ChartStudio

Chartstudio je webová aplikace přímo od vývojářů knihovny Plotly zmiňované v kapitole 2.3 a je na ní hned znát, že pochází z rukou zkušených vývojářů. Na svých stránkách je dokonce zmíněno, že Chartstudio využívají velké společnosti jako je Tesla nebo The Washington Post.[16]

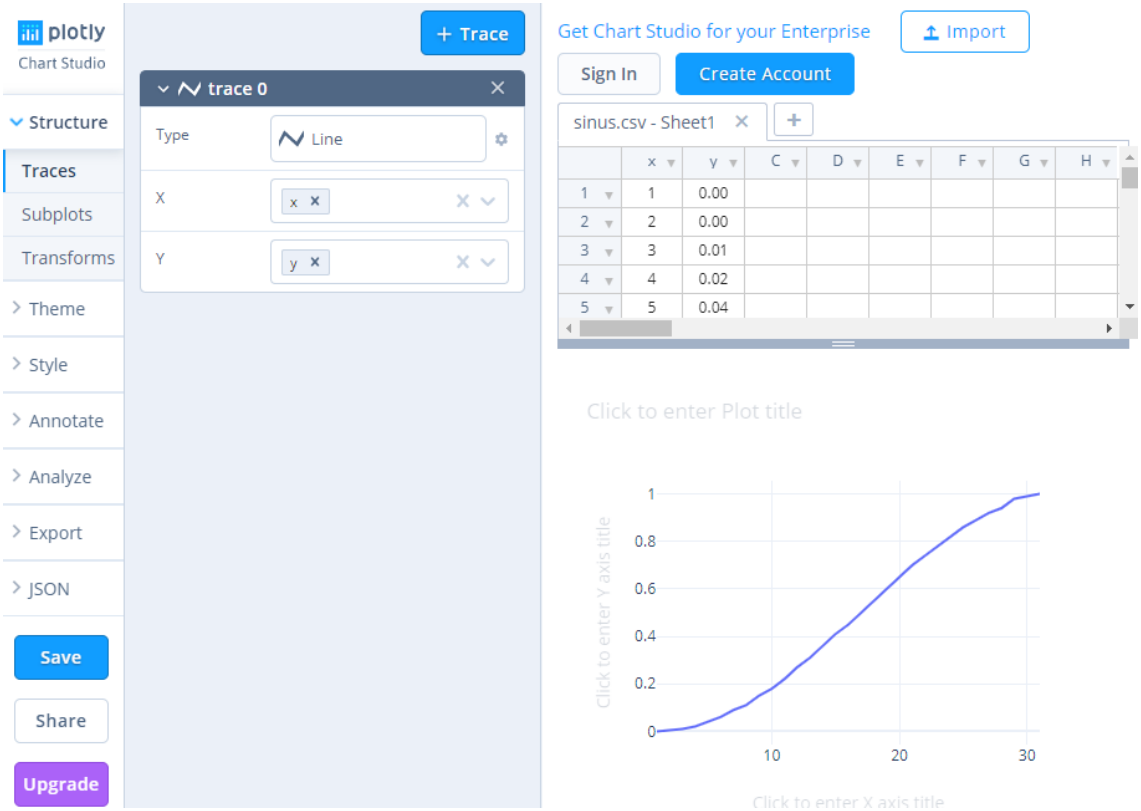
Chartstudio není pouze frontendová aplikace, ale celé komplexní řešení poskytující, jak frontendový editor, tak webové API. Vývojáři cílí na širokou použitelnost, a tak nabízejí propojení s jazyky pro zpracování dat jako je Python, R či MATLAB. Pro programátory webových aplikací dokonce nabízí propojení aplikací využívající knihovnu Plotly s Chartstudiem jediným řádkem kódu.

Frontendové prostředí jako takové působí poměrně složitě kvůli nic neříkajícím záložkám, kterými se uživatel musí proklikat, aby zjistil, kde se co nastavuje. Poté co se ale uživatel s prostředím naučí, tak zjistí, že Chartstudio nabízí velké množství možností, jak pracovat s vloženými daty.[17] Umožňuje tvorbu více samostatných grafů, které mohou obsahovat hned několik nezávislých hodnot najednou. V grafu lze jednoduše vybírat pouze úseky, které uživatele zajímají. Dokonce aplikace umožňuje data filtrovat a provádět nad nimi různé výpočty a operace.

Na obrázku číslo 3.2 lze vidět uživatelské rozhraní ChartStudia. Pro práci je potřeba nejdříve v pravém horním rohu kliknout na tlačítko Import a nahrát do aplikace data. Tím se zobrazí tabulka všech hodnot, která je u složitých časových řad poměrně nepřehledná. Dále je potřeba zvolit jaké hodnoty se mají vykreslit do grafu. To se udělá kliknutím na tlačítko +Trace. Uživatel zvolí názvy sloupců, které obsahují chtěné hodnoty, vybere styl grafu a Chartstudio zobrazí výsledek. V levém menu se nachází další nastavení a operace, které lze nad daty a grafy provádět.

Skoro by to vypadalo, že až na pro mě subjektivně zmatené uživatelské prostředí, je Chartstudio dokonalé pro potřeby práce, a hlavně pro potřeby výzkumu Doprava 2020+. Bohužel ale Chartstudio není aplikace vhodná k použití zdarma při rozsáhlejších projektech a výzkumech jako by mohl být právě výzkum zmiňovaný dříve. Její cenový model je nastaven tak, že verze zdarma přináší nemalé omezení například v omezení velikosti vstupních dat na 500KB, nutnosti spoléhat se na Cloudové prostředí poskytované vývojáři. Aplikace má dokonce limit kolik lidí si může zobrazit výsledný graf.

Aplikace je ale důkazem, že použité technologie na její tvorbu jsou schopné vytvořit aplikaci, která elegantně zvládá rozsáhlá data časových řad a dokáže s nimi velmi rychle pracovat.



Obrázek 3.2: Ukázka prostředí Plotly - ChartStudio

4 Návrh vlastního řešení

Tato kapitola popisuje, jaké jsou požadavky na webovou aplikaci, která vyplývá z požadavků výzkumného týmu pracujícím na projektu Doprava 2020+, která mi byla tlumočena vedoucím práce. Na základě požadavků je zde popsáno, jaké technologie byly vybrány k tvorbě webové aplikace, aby jim bylo všem vyhověno. Dále je zde popsána struktura aplikace, která je zobrazena na potřebných diagramech společně s jejich popisy a komentáři k nim.

4.1 Požadavky na aplikaci

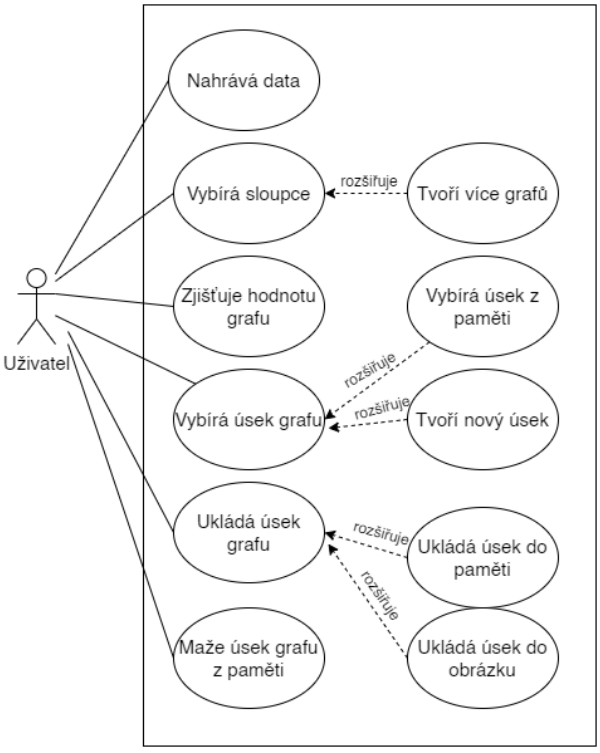
Účel aplikace

- Aplikace bude sloužit k zobrazování číselných dat do 2D grafů.
- Aplikace bude zpracovaná jako webová aplikace.
- Data budou přijímána z csv souborů.
- Uživatel bude mít možnost vybrat, které hodnoty chce zobrazit do grafů.
- Uživatel bude mít možnost pozměnit vybraná data nějakou z vytvořených funkcí (například klouzavým průměrem).
- Aplikace bude umožňovat buď tvorbu jednoho samotného grafu, nebo více jak jednoho podle potřeb uživatele.
- Grafy budou mít možnost zobrazovat více veličin najednou, pokud budou mít stejné hodnoty osy X.
- V grafech bude možnost vybírat a studovat pouze jejich části zvolené uživatelem.
- Vybrané části grafu bude možnost ukládat a bude možnost je zobrazit později.
- Uživatel bude mít možnost zjistit okamžitou hodnotu nebo její nejbližší hodnotu v kterékoliv části grafu.
- Uživatel bude mít možnost exportovat aktuální úsek grafu do vektorové nebo rastrové grafiky.

Omezení a specifikace

- Aplikace bude potřebovat ke svému chodu připojení k internetu, není tak zaručena plná funkčnost při výpadku připojení.
- Aplikace bude ke svému chodu využívat desktopové verze internetových prohlížečů, není tak zaručena funkčnost na mobilních a jiných zařízeních.
- Aplikace bude vyvíjena na nejnovějších dostupných technologiích, nebude tak zaručena funkčnost na neaktuálních prohlížečích a systémech.
- Aplikace bude vyvíjena na stolním počítači s 64bitovým operačním systémem Windows 10, kde je dostupných minimálně 8 GB operační paměti, není tak zaručena funkčnost na zařízeních s nižším počtem operační paměti nebo jiném operačním systému.

Případy užití



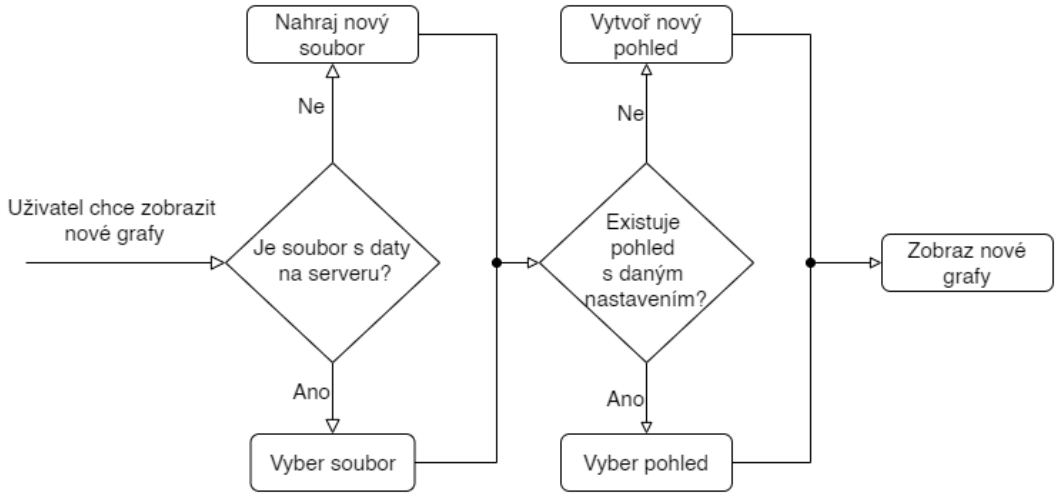
Obrázek 4.1: Případy užití aplikace

Aplikace nemá za cíl nijak řešit role uživatelů ani jinak uživatele dělit, tudíž z pohledu aplikace jsou všichni, kteří budou s aplikací pracovat, uživatelé. To je zachyceno v UseCase diagramu na obrázku 4.1 jediným actorem. Práce uživatele s aplikací by se dala rozdělit do dvou pod úloh: tvorba grafů a následná práce s grafy. Tvorbě grafů připadají první dva případy užití shora z UseCase diagramu. Proces

tvorby grafu bude popsán dále, jednoduše ale uživatel poskytne aplikaci data, tím že je nahraje a poté si zvolí sloupce, které chce vykreslit do grafů. S volbou sloupců je úzce spjato i další nastavení grafu, případně volba funkce nad vybranými daty. Uživatel má možnost vytvořit i více grafů zároveň, toto je zachyceno v UseCase diagramu jako rozšiřující akce.

Proces práce s grafy umožňuje uživateli více akcí než nastavovací proces. Akce pro práce s grafy jsou všechny zbývající akce z UseCase diagramu. Když uživatel pracuje s grafy, tak má možnost zjišťovat aktuální hodnotu grafu v libovolném bodě nebo její aproximaci nebo alespoň její nejbližší hodnotu. Dále má uživatel možnost zobrazit jen část grafu. Toho uživatel docílí buď vybráním již uloženého úseku nebo tak, že si úsek zvolí sám. Úsek poté může uložit buď pro pozdější použití do paměti, aby se k němu šlo vrátit, anebo si z něj uloží obrázek do úložiště svého zařízení. V poslední řadě poté má uživatel možnost mazat uložené úseky z paměti.

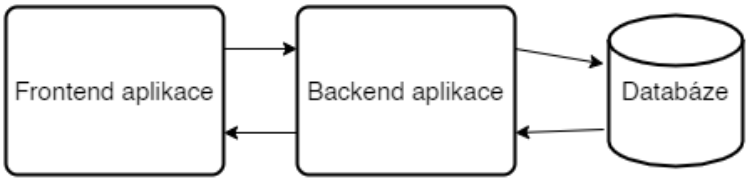
Na rozdíl od práce s grafem, kdy kterákoliv akce může nastat v libovolný čas a akce mohou přicházet v libovolném pořadí, tak u nastavovacího procesu tomu tak není. Jak lze vidět na obrázku 4.2, tak nastavovací proces se dá zakreslit jako jednoduchý postup, kdy možné akce závisí na stavu dat na serveru a toho, jaký graf chce sestrojit. Jako první se uživatel musí rozhodnout, jestli chce pracovat s daty, která byla již nahrána dříve anebo chce pracovat s daty novými. Nová data musí uživatel do aplikace nejdříve nahrát a poté je ve stejném stavu, jako by si zvolil již existující data. Dále přichází volba hodnot os grafu a další nastavení, která jsou označená jediným slovem „pohled“. Pro dříve nahraná data si aplikace pamatuje vytvořené pohledy, a tak si uživatel může nechat zobrazit data z dříve vytvořených pohledů, nebo si musí vytvořit pohled nový.



Obrázek 4.2: Diagram nastavovacího procesu

4.2 Návrh struktury aplikace

Z případu užití vyplívá, že aplikace se bude skládat ze tří hlavních částí. První bude frontendová stránka běžící ve webovém prohlížeči, skrz kterou bude uživatel komunikovat s druhou částí, kterou tvoří backendová aplikace běžící na serveru. Backend bude ke svojí činnosti uchovávat data v databázi. Backend tedy po požadavku z frontendové aplikace vezme data z databáze, a pokud to bude třeba, tak je zpracuje a vrátí je do frontendové aplikace. Tato celá komunikace je znázorněna pro přehlednost na obrázku 4.3.



Obrázek 4.3: Základní struktura aplikace

4.2.1 Návrh databáze

V databázi bude potřeba ukládat nahrané soubory, kvůli provázání s pohledy a uloženými úseky grafu. Tím se předejde zbytečnému provozu, kdy by se jinak musely soubory opakovaně nahrávat ke zpracování. Takto se celá komunikace urychlí.

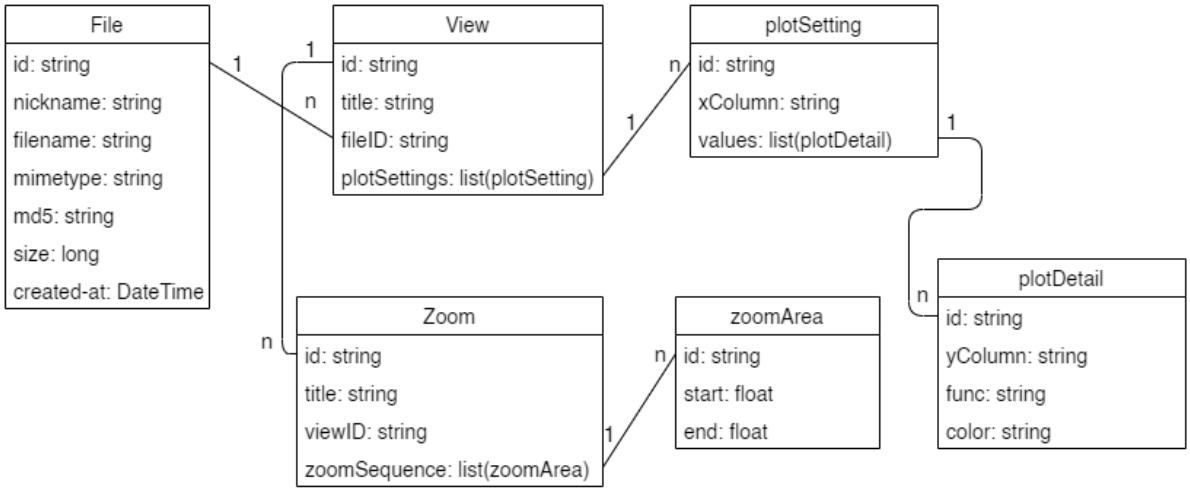
Dále bude potřeba ukládat samotné pohledy. Pohledy reprezentují nastavení grafů zobrazující data ze souboru. Každý pohled popisuje z kolika grafů se skládá, jaké hodnoty budou mít grafy na svých osách, jaké funkce se budou provádět nad daty a dodatečné nastavení pro odlišení jednotlivých průběhů v grafu od sebe, jako je barva průběhu.

Finálně bude potřeba ukládat úseky grafu vybrané uživatelem. Ty se skládají z názvu a následně sekvence výběrů. Sekvence je to z toho důvodu, pokud by šlo o nějaký konkrétnější a detailnější úsek, který vznikl tak, že uživatel vybral úsek z úseku, a následně z toho úseku vybral další úsek atd.

Pro abstrakci od konkrétního databázového přístupu jsem zakreslil komponenty, které bude potřeba uchovávat v databázi jako diagram tříd. Tento diagram lze vidět na obrázku 4.4.

Třída *File* slouží k uchování informací o nahraném souboru na serveru. Soubor má o sobě uložený název, díky kterému lze získat soubor ze serverového úložiště. Dále se ukládá uživatelský název *nickname*, který slouží k orientaci mezi soubory v uživatelském prostředí. *Mimetype* slouží pro kontrolu, zda se jedná opravdu o validní csv soubor, popřípadě pokud by byla implementována podpora jiných souborů než jen csv souborů, tak podle tohoto atributu lze volit typ zpracování.

Pokud by byl na server nahrán soubor, který odpovídá nějakému souboru, který byl nahrán dříve, tak lze porovnat hodnoty MD5 hashů obou souborů, a lze na duplicitu přijít a neukládat tak stejný soubor dvakrát. Proto se bude ukládat právě hodnota MD5 funkce. Pro zjištění identity můžeme ještě k MD5 hodnotě porovnávat



Obrázek 4.4: Databázové komponenty

velikost souboru, proto je vhodné ji také ukládat. Jako poslední atribut pro třídu *File* je časový údaj o tom, kdy byl soubor nahrán na server.

Druhá komponenta představuje pohled nad daty a je označena jako *View*. Tato třída o sobě uchovává uživatelský název pro snadné odlišení pohledů od sebe. Každý pohled přísluší právě jednomu uloženému souboru, ale každý soubor může mít k sobě neomezeně pohledů. Mezi těmito komponentami je tedy vazba 1:N, kdy každý pohled má uloženo ID souboru, ke kterému přísluší. Jelikož každý pohled může představovat různé množství nezávislých grafů nad daty stejného souboru, tak třída má seznam, ve kterém jsou uložené všechny nastavení grafů pojmenované jako *plotSettings*. Mezi pohledy a nastaveními grafů se tady nachází vazba 1:N, kdy každý pohled může mít více grafů, ale konfigurace grafu patří pouze k jednomu konkrétnímu pohledu.

Všechny hodnoty grafu musí sdílet stejnou hodnotu osy X, aby šly vykreslit do stejného grafu. Třída *plotSetting*, která reprezentuje nastavení jednotlivého grafu, tak ukládá název sloupce z dat, který slouží jako hodnota dat v ose X. Hodnoty osy Y se poté od sebe liší a jsou tak reprezentovány další komponentou. PlotSetting má ale s touto komponentou vazbu 1:N, kdy každé nastavení grafu může mít teoreticky neomezeně řad hodnot Y, ale tato nastavení řad přísluší jen k jednomu jedinému grafu.

Komponenta, která popisuje nastavení hodnot Y grafu, se nazývá *plotDetail* a uchovává informaci o názvu sloupce z nahraných dat, který určuje, jaká data se mají pro hodnoty Y použít. Dále se ukládá informace o funkci, která se má nad daty provést. Pokud se žádná funkce použít nemá, hodnota zde bude prázdný řetězec. Pro rozlišení hodnot od sebe v grafu se ukládá i barva, kterou se má řada obarvit při vykreslování grafu.

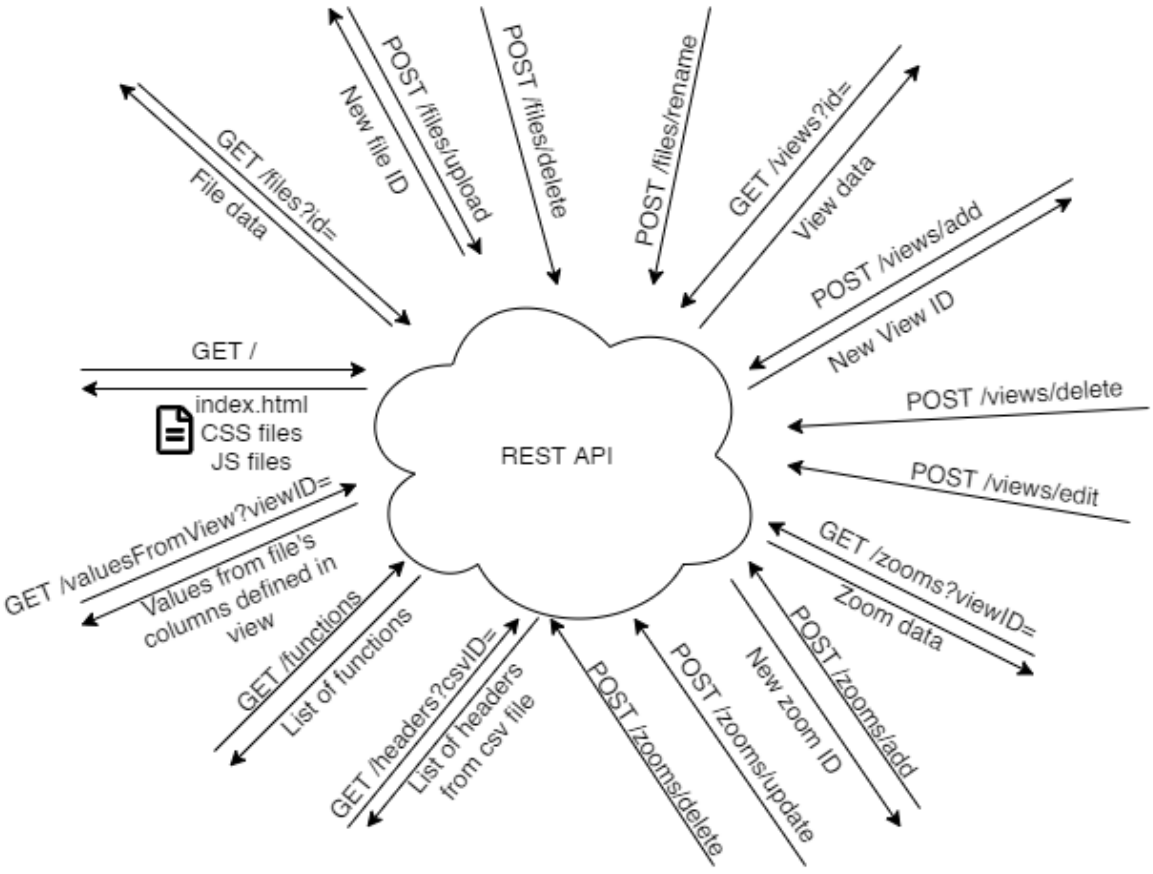
Komponenta *Zoom*, uchovává informace o uloženém úseku grafu. S grafem je provázána vazbou 1:N, kde každý *Zoom* přísluší jen k jednomu konkrétnímu pohledu, nicméně pohledy mohou mít neomezeně uložených úseků. Pro snadné rozlišení úseků se ukládá jeho název nastavený uživateli. Jak bylo napsáno dříve, tak finál-

ní úsek se může skládat z více úseků, kterou jsou skládané do sebe. Třída *Zoom* proto obsahuje seřazený seznam těchto úseků, které jsou v databázi reprezentovány komponentou *zoomArea*. Každý úsek je pak určen hodnotou začátku a konce úseku, proto se u komponenty *zoomArea* uchovávají jen tyto hodnoty.

4.2.2 Návrh serverového API

Když je určeno, co vše se bude v databázi ukládat a je dán přehled o tom, co vše by měla backendová aplikace umožňovat, je na místě navrhnout jaké všechny přístupové body, anglicky *endpoints*, bude aplikace poskytovat. Přístupové body jsou adresy, kam když přijde správný HTTP požadavek s příslušnými daty, tak se na serveru provedou určité akce a server vrátí výsledek.

První, velice důležitý přístupový bod je root, který odpovídá jen adrese serveru. Pokud přijde GET požadavek na tento bod, například z prohlížeče zadáním adresy, tak aplikace pošle klientovi celou frontend aplikaci. Celá aplikace znamená všechny html, css, js a další potřebné soubory.



Obrázek 4.5: Schéma koncových bodů API

Další přístupové body se odvíjí od komponent databáze, protože bude potřeba poskytovat všechny CRUD operace v nějakém smyslu, pro manipulaci s daty v databázi, prostřednictvím API. CRUD je zkratka, která stojí za základními operacemi s daty. C značí create, neboli vytvořit nový záznam v databázi. R stojí za slovem read, tedy čtení informací o komponentě z databáze. U znamená update, tudíž operaci, která dovoluje měnit existující záznamy v databázi, a jako poslední písmenko zkratky je D, které značí slovo delete, neboli odstranění záznamu z databáze.

Jak lze vidět na obrázku 4.5, tak přístupových bodů je potřeba alespoň 16. Pokud se jedná právě o CRUD operace pro manipulaci s daty v databázi, tak struktura požadavku je *adresa_server/název_komponenty/název_akce*. Prakticky stačí jen operace pro komponenty *File*, *View* a *Zoom*. Ostatní komponenty, které patří pod tyto základní komponenty, lze měnit a pracovat s nimi v rámci zmíněných hlavních třech komponent.

Mimo CRUD operací a dotazu na data frontend aplikace, jsou potřeba ještě další tři přístupové body. První je pro získání dostupných funkcí nad daty, ten se nachází na adrese *adresa_serveru/functions* a jedná se o GET požadavek. Druhý je pro získání hlaviček z csv souboru. Jedná se opět o GET request tentokrát na adresu *adresa_serveru/headers* s parametrem *csvID*, který odpovídá ID souboru z databáze. Poslední přístupový slouží k získání dat pro tvorbu grafů na základě vybraného pohledu. Jedná se o GET request na adresu *adresa_serveru/valueFromView*. Všechny koncové body jsou pro přehlednost vypsány do tabulky 4.1

Tabulka 4.1: Přístupové body backend aplikace

Přístupový bod	Typ dotazu	Parametry/data	Popis
/	GET	-	Slouží k získání frontend aplikace.
/files	GET	id: string	Bez parametru slouží k získání ID všech souborů v databázi, s parametrem vrací všechny informace o daném souboru.
/files/upload	POST	files: form-data	Uloží poslané soubory, jestli vyhovují, na serverové úložiště a do databáze, a vrátí jejich ID.
/files/delete	POST	fileID: string	Uloží poslané soubory na serveru a do databáze, jestli vyhovují, a vrátí jejich ID.
/files/rename	POST	fileID: string, nickname: string	Změní uživatelské jméno souboru podle ID na hodnotu nickname.
/views	GET	id: string, fileID: string	Bez parametru slouží k získání ID všech pohledů uložených v databázi, jinak vrátí všechny dostupné informace o pohledu podle jeho ID společně s provázanými hodnotami plotSetting a *plotDetail*. Pokud je součástí dotazu fileID, tak vrátí ID všech pohledů, které přísluší danému souboru.
/views/add	POST	fileID: string, plotSettings: list(Object), title: string	Uloží do databáze nový pohled z údajů těla požadavku. Po vytvoření vrátí ID nového pohledu.
/views/delete	POST	id: string	Odstraní z databáze pohled podle ID pohledu, a společně s pohledem odstraní i všechny související *plotSetting* a *plotDetail* záznamy.
/views/edit	POST	viewID: string, fileID: string, title: string, plotSettings: list(Object)	Změní údaje pohledu v databázi podle hodnot v těle dotazu. *ViewID* je povinná součást těla dotazu, zbytek je volitelný podle toho, co chce uživatel změnit.
/zooms	GET	viewID: string	Slouží k získání ID všech úseků, které jsou uložené pro pohled daný jeho ID v parametrech dotazu.

/zooms/add	POST	title: string, viewID: string, sequence: list(list(float))	Uloží do databáze nový úsek podle údajů v těle dotazu. Hodnota atributu dotazu *sequence* musí vyhovat struktuře komponenty *zoomArea* v databázi. Musí se tedy jednat o pole polí které mají dvě číselné hodnoty určující počátek a konec výběru.
/zooms/update	POST	zoomID: string, viewID: string, title: string, zoomSequence: list(list(float))	Změní hodnoty úseku v databázi podle ID úseku *zoomID* a dalších hodnot v těle dotazu. *ZoomID* je povinné, další atributy jsou volitelné podle toho, co je potřeba změnit.
/zooms/delete	POST	id: string	Odstraní záznam o úseku grafu z databáze podle ID. Zároveň odstraní i všechny komponenty zoomArea provázané s mazaným úsekem
/headers	GET	csvID: string	Získá všechny hlavičky csv souboru, podle ID souboru, a pošle je zpátky jako seznam.
/functions	GET	-	Získá všechny implementované funkce pro data a jejich názvy pošle zpátky jako seznam.
/valueFromView	GET	viewID: string	Slouží k získání finálních dat pro vykreslení grafů. Podle ID pohledu načte soubor a pro všechny *plotSetting* načte potřebná data, provede potřebné operace a data vrátí ve slovníkové podobě.

4.3 Technologie využité ve vlastním řešení

Pro tvorbu frontendové aplikace byla vybrána knihovna D3.js, která byla popsána v kapitole 2.2. Důvodem bylo to, jaké všechny možnosti používání poskytuje. Díky D3.js lze implementovat všechny požadavky na aplikaci a zároveň pořád nabízí široké možnosti dalšího vývoje za použití jedné knihovny. Ostatní knihovny zmíněné v kapitole 2 cílí spíše na konkrétní implementaci vizualizace dat, zatímco vývojář s D3.js má volnou ruku a zároveň je knihovna přínosná i pro jiné funkcionality na webu než jen vizualizace dat.

Dále byl pro vývoj použit program Webpack, který slouží k balení veškerých souborů, které programátor vytvoří, do balíčků podle programovacího jazyka pro snadnou distribuci výsledné aplikace. Používání Webpacku programátora nutí rozdělovat kód do modulů, což vede na přehlednější a snadněji rozšiřitelnou aplikaci.

Backendová část aplikace byla vytvořena na prostředí Node.js s frameworkem Express.js. Celá aplikace je tak programovaná v jazyce Javascript. Použití Javascriptu na serveru umožňuje používat asynchronní programování, které umožňuje serveru rychleji zpracovávat dotazy bez čekání na data pro jiné dotazy z jiných služeb, jako je databáze nebo souborový systém. Vzhledem k tomu, že primární úloha backendové aplikace je zpracovávat data právě z databáze a souborového systému, tak je asynchronní přístup velkým přínosem.

Pro uchovávání dat byla použita databáze MongoDB. MongoDB je databáze typu NoSQL a ukládá data do takzvaných dokumentů, se kterými se pracuje stejně jako s javascriptovými objekty. Jelikož MongoDB samo o sobě dovoluje ukládat do dokumentů libovolná data s libovolnou strukturou, tak programátor musí hlídat, jaká data jak a kam ukládá. Použití knihovny Mongoose obchází tento problém a nabízí možnost vytvářet si šablony dokumentů, a vkládaná data musí poté vyhovovat této šabloně. Tím se zamezí chybám s nekonzistentními daty mezi stejnými dokumenty.

Následující tabulka 4.2 obsahuje údaje o verzích knihoven využívané při vývoji.

Tabulka 4.2: Verze použitých knihoven

Název knihovny/technologie	Verze
D3.js	6.3.1
Webpack	5.10.0
Express.js	4.16.1
Mongoose	5.12.0
MongoDB	4.4.4

5 Tvorba aplikace pro vizualizaci dat

5.1 Tvorba backend aplikace

Struktura souborů

- .env – soubor, ve kterém jsou uložené hodnoty pro běhové prostředí
- app.js – hlavní modul serveru
- /bin
  - www.js – vstupní bod aplikace
- /dataFunctions – adresář pro funkce operací nad daty
  - dataFunctions.js – hlavní modul, který poskytuje funkce operací nad daty
- /dbSchemas – adresář pro šablony dokumentů knihovny Mongoose
  - fileSchema.js – šablona dokumentu File
  - viewSchema.js – šablona dokumentu View
  - zoomSchema.js – šablona dokumentu Zoom
- /files – adresář pro ukládání nahraných souborů s daty k vizualizaci
- /models – adresář tříd modelů
  - CsvModel.js – Model obsahující funkce pro práci s CSV soubory
  - FileModel.js – Model obsahující funkce pro práci se soubory v databázi
  - ViewModel.js – Model obsahující funkce pro práci s pohledy grafu
  - ZoomModel.js – Model obsahující funkce pro práci s úseky grafu
- /node_modules – adresář knihoven z npm
- package_json – soubor s výčtem všech npm modulů a knihoven
- /public – adresář, který obsahuje výslednou frontend aplikaci
  - /dist – výstup z Webpacku

- /img – složka s obrázky použitými na webu
- /src – složka pro externí skripty a knihovny
- index.html – HTML dokument single page aplikace

— /routes – adresář tříd zpracovávající koncové body API

- fileRouter.js – třída koncových bodů /files
- index.js – třída nezařazených koncových bodů
- viewRouter.js – třída koncových bodů /views
- zoomRouter.js – třída koncových bodů /zooms

5.1.1 Soubor App.js

Soubor app.js je jedním z nejdůležitějších souborů celé aplikace. Soubor obsahuje modul serverové aplikace, která je spouštěna souborem www.js ve složce /bin. V app.js se spravuje nastavení celé serverové aplikace. Na prvních řádcích tohoto souboru se načítají všechny moduly, které má server používat. Moduly buď mohou představovat propojené knihovny, nebo vlastní moduly, jako například routery, moduly zpracovávající http požadavky na konkrétní koncové body aplikace.

Dále je v souboru app.js řešeno nastavení použitých modulů, a jelikož takřka veškeré služby jsou řešeny jako moduly, tak jeden z modulů řeší i připojení k databázi. Kód pro připojení k databázi lze vidět na kódu 1. Jelikož je používání databáze řešeno knihovnou Mongoose, tak připojení je otázka jediné funkce z knihovny. Na prvním řádku se načítá URI databáze. Ta je uložená v souboru .env pod klíčem DB_URI. Práci s hodnotami v tomto souboru zajišťuje knihovna Dotenv. Na druhém řádku je poté použita funkce *connect* z modulu mongoose. Jako první parametr se uvádí právě URI serveru, na kterém běží MongoDB databáze, a dále další nastavení připojení. *UseNewUrlParser* povoluje modulu využívat vlastní ovladač ke zpracování řetězců z MongoDB a druhý parametr *useUnifiedTopology* povoluje modulu využívat nejnovější způsob autorů knihovny zacházení s připojením k databázovým serverům. Pro přímý přístup k databázi slouží poté reference uložená v proměnné *db*.

```
const dbURI = process.env.DB_URI;
mongoose.connect(
  dbURI,
  {useNewUrlParser: true, useUnifiedTopology: true}
  ).then();
const db = mongoose.connection
```

Kód 1: Vytvoření databázového připojení

Express.js dovoluje přidělit koncové body, respektive cesty k nim, modulům zvané routery. To dovoluje lépe organizovat kód aplikace, protože kód zpracování různých požadavků lze rozdělit do vlastních modulů. Každý modul routeru je instance

třídy Router z frameworku Express. V této aplikaci se nachází celkem čtyři routery: *fileRouter*, *indexRouter*, *viewRouter* a *zoomRouter*. Propojení routerů s aplikací se dělá právě v modulu serverové aplikace app.js. V kódu 2 lze vidět připojení routerů s cestami na serveru. Je použit příkaz *use* hlavního modulu frameworku Express *app*. Jako první parametr se uvádí cesta v adrese serveru, kterou má router zpracovávat. Pokud je uvedena jako cesta hodnota /zooms, tak tento router bude zpracovávat všechny požadavky, které začínají adresou *adresa_serveru/zooms/*.

```
app.use('/', indexRouter);
app.use('/zooms', zoomRouter);
app.use('/views', viewRouter);
app.use('/files', fileRouter);
```

Kód 2: Použití routerů

5.1.2 Komunikace s databází

Aplikace využívá non-SQL databázi MongoDB a všechny funkcionality pro práci s ní poskytuje knihovna Mongoose. Připojení k databázovému serveru bylo ukázáno v kódu 1, nicméně to k práci s databází nestačí. Je potřeba si ještě definovat šablony pro dokumenty, které se budou v databázi objevovat. Tyto šablony se vytváří jako moduly, které jsou instancí třídy *Schema* z knihovny Mongoose. V kódu 3 je vidět definice potřebných dokumentů pro uložení všech informací pohledu nad daty. Každá šablona je instance třídy Schéma, která jako parametr přijímá slovník hodnot, které popisují ukládané informace. Formát je takový, že klíč se použije jako název hodnoty a hodnota klíče jako jeho datový typ. Jak lze vidět u šablon *plotSettingsSchema* a *viewSchema*, tak jako datové typy lze využít i jiné definované šablony. *CourseValuesSchema* si bude ukládat tři hodnoty, které jsou typu řetězec a nazývají se *yColumn*, *func* a *color*. *PlotSettingsSchema* bude ukládat pole objektů, které odpovídají hodnotám definovaným v *CourseValuesSchema*, a přidá k němu textový řetězec s názvem *xColumn*.

```
const courseValuesSchema = new Schema({
  yColumn:  { type: String},
  func:     { type: String},
  color:    { type: String}
});
const plotSettingsSchema = new Schema({
  xColumn:  { type: String },
  values:  { type: [courseValuesSchema] }
});
const viewSchema = new Schema({
  title:        { type: String },
  fileID:       { type: String },
  plotSettings: { type: [plotSettingsSchema] }
});
```

Kód 3: Šablony Mongoose dokumentů

Moduly takto vytvořených šablon je jen potřeba načíst v místě, kde se bude pracovat s databází, nejčastěji v modelech. Modul buď obsahuje statické metody, které slouží k hledání v databázi, úpravám, mazání atd. Pokud je ale potřeba přidat do databáze nový záznam, tak je potřeba vytvořit instanci šablony dokumentu. V kódu 4, který pochází ze souboru ViewModel.js je vidět úsek kódu, kde se nejdříve načte modul, který obsahuje šablonu pohledu. Na dalších řádcích je poté vidět kód přidávající nový pohled do databáze. Nejdříve se vytvoří instance šablony, která přijímá slovník hodnot jako parametr. Poté se provede funkce validace, která zkontroluje, zda hodnoty odpovídají předpisu dokumentu ze šablony, a pokud ano, tak zavolá funkci *save*, která pohled zapíše do databáze. Obě operace jsou asynchronní, což poskytuje možnosti reagovat na kladný i záporný výsledek operace, a zároveň celá aplikace nestojí při čekání na odpověď z databázového serveru.

```
const View = mongoose.model('Views', viewSchema);
...
const view = new View({
  title: title,
  fileID: fileID,
  plotSettings: plotSettings
});
view.validate().then(
  () => {
    view.save().then((newView) => {
...
```

Kód 4: Přidání záznamu do databáze

5.1.3 Zpracování HTTP dotazů

Webová aplikace sice slouží primárně jako REST API pro frontendovou aplikaci, ale i tak se snaží přiblížit architekturou běžné webové aplikaci, která využívá MVC strukturu. Konkrétně tato aplikace dělí kód hlavně do M a C, tedy modelů a kontrolerů. Jak již bylo zmíněno dříve, Express.js dovoluje přesměrovávat dotazy pomocí routerů, a routery v podstatě splňují definici kontroleru. Převezmou http požadavky, získají potřebná data z modelů, a vrátí uživateli odpověď. Data od uživatele i pro uživatele se přenáší ve formátu JSON. Pokud by tedy měla být nějaká vrstva označena jako View, pak to bude právě parser JSON řetězců.

V aplikaci se nachází celkem čtyři routery. *ViewRouter*, který zpracovává dotazy na adresu /views, *fileRouter*, který zpracovává dotazy na adresu /files, *zoomRouter*, který zpracovává dotazy na adresu /zooms, a jako poslední *indexRouter*, který zpracovává vše ostatní.

Jak se konkrétní routery v aplikaci aktivují, aby jim byly předávány dotazy, bylo ukázáno v kódu 2 v souboru app.js. Konkrétní soubory obsahující kódy konkrétních routerů. Příklad definice routeru je v kódu 5. Nejdříve je potřeba importovat knihovnu Express.js. Poté se vytvoří objekt typu *Router*. Poté následuje definice popsaná dále, a nakonec je potřeba vytvořený objekt routeru exportovat jako modul, aby ho bylo možné použít právě v app.js souboru.

```
const express = require('express');
const router = express.Router();
...
module.exports = router;
```

Kód 5: Vytvoření routeru

Ke zpracování dotazů má router k dispozici řadu metod, které odpovídají možným typům http požadavků. Je-li potřeba vytvořit koncový bod, který bude zpracovávat požadavek typu GET, použije se metoda *router.get()*, pokud bude požadavek typu POST, použije se metoda *router.post()*.

Se všemi těmito metodami se pracuje velice podobně. Jako první parametr se uvádí adresa koncového bodu. Výsledná adresa se spojuje s cestou, kterou má router k sobě přiřazenou. Je-li například definována v metodě *post* adresa /add, a pracuje-li se například ve *viewRouteru*, který má v app.js uvedenou adresu /views, tak tato metoda bude zpracovávat výhradně jen dotaz na adresu *adresa_serveru/views/add* typu POST.

Jako ukázka je přesně tato metoda ukázána v kódu 6. Jako druhý parametr funkcí routeru je callback funkce, která zpracovává požadavek. Tato funkce získává jako parametry objekt reprezentující požadavek a objekt, reprezentující odpověď. Z požadavku je potřeba nejdříve získat data, která se nachází v atributu *body*, jestli se jedná o požadavek, který má definované tělo, popřípadě jsou data v atributu *query*, pokud má požadavek data jako součást URL.

Po zpracování dat je zavolána funkce modelu, která se stará o přidání nového pohledu do databáze. Podle výsledku akce modelu je nastaven kód odpovědi a jsou k ní přidaná konkrétní data. V tomto případě ID nového pohledu, pokud je přidání úspěšné, a pokud není, tak se pošle zpět chybová hláška. Na tomto principu pracují i všechny ostatní routery v aplikaci.

```
router.post('/add', ((req, res) => {
  try {
    const fileID = req.body.fileID;
    const title = req.body.title;
    const plotSettings = JSON.parse(req.body.plotSettings);
    viewModel.addView(title, fileID, plotSettings).then(
      (id) => { res.status(200).send(id); },
      (err) => { res.status(400).send(err); }
    );
  } catch (ex) {
    res.status(400).send(ex.message);
  }
}));
```

Kód 6: Zpracování dotazu na adresu /views/add

5.1.4 Modely serverové aplikace

Jako model se označuje část aplikace, která získává a zpracovává data, často z nějakého úložiště. V aplikaci se nachází čtyři modely. První je *CsvModel.js*, který čte data z uložených CSV souborů v úložišti. Druhý je *FileModel.js*, který pracuje s dokumenty *File* v databázi. Zároveň se při nahrávání souboru na server stará o jeho uložení a následné přidání do databáze. Třetí je *ViewModel.js*, který pracuje s jednotlivými pohledy neboli dokumenty *View* v databázi. Poslední model je *zoomModel*, který pracuje s výseky grafu.

Jako model by se i dal označit modul s funkcemi pro načtená data. Tento modul je uložen v souboru *dataFunctions.js* ve složce dataFunctions/. Tento modul zprostředkovává funkce, která mění načtená data. Aktuálně jsou zde definované funkce počítající klouzavé průměry, pro různé rozsahy hodnot, a funkce, která zvětší hodnotu dat na ose Y. Pokud se budou v budoucnu přidávat další funkce, tak musí na ně být odkaz v tomto souboru.

Veškeré funkce a metody standartních modelů v této aplikaci využívají asynchronní programování. V aplikaci tak lze jednoduše reagovat a pracovat s informací, jestli akce skončila podle představ, popřípadě reagovat na chybu. Zároveň aplikace může zpracovávat jiné požadavky, zatímco se čeká na dokončení operace modelu.

5.1.5 Dokumentace funkcí modelu

CsvModel.getHeaders
Slouží k získání názvů sloupců z csv souboru.
Parametry:
filename: String - název souboru uloženého na serveru
Návratová hodnota:
Promise<List[]> - při úspěšném načtení souboru vrátí seznam řetězců, které odpovídají názvům sloupců v CSV souboru

CsvModel.getValueTuples
Slouží k získání dvojic x a y z csv souboru podle názvů sloupců.
Parametry:
filename: String - název souboru uloženého na serveru
xValue: String - název sloupce pro hodnoty x
yValue: String - název sloupce pro hodnoty y
Návratová hodnota:
Promise<List[]> - při úspěšném načtení souboru vrátí pole objektů, které mají atributy x a y, s hodnotami z csv souboru

FileModel.saveCSV
Uloží do úložiště na serveru a do databáze soubor poslaný http požadavkem.
Funkce zároveň kontroluje, jestli je soubor opravdu typu CSV.
Parametry:
inFile: File - objekt souboru, který se má uložit
Návratová hodnota:
Promise<String|null> - při úspěšném uložení souboru vrátí jeho ID z databáze, pokud soubor neprojde validací, je odmítnut

FileModel.getFiles
Slouží k získání všech uložených souborů v databázi.
Pokud narazí na soubor, který je v úložišti a ne v databázi, tak ho smaže.
Návratová hodnota:
Promise<Object[]> - seznam objektů reprezentující soubory v databázi

FileModel.getFileByID
Najde soubor v databázi podle ID a vrátí ho.
Parametry:
fileID: String - ID souboru v databázi
Návratová hodnota:
Promise<Object[]> - objekt nalezeného souboru z databáze

FileModel.findSimilarFile
Pokusí se najít soubor se stejnou hash hodnotou MD5 funkce a stejným typem souboru.
Parametry:
md5: String - Unikátní hash souboru
mimetype: String - Typ souboru
Návratová hodnota:
Promise<Object[]> - objekt nalezeného souboru z databáze

FileModel.deleteFile
Smaže soubor podle ID z databáze i z úložiště.
Parametry:
fileID: String - ID souboru v databázi
Návratová hodnota:
Promise<null|String> - funkce nic nevrací pokud byl soubor úspěšně smazán, vrací ale chybovou hlášku, pokud nastane problém

FileModel.changeNickname
Změní uživatelské jméno souboru.
Parametry:
fileID: String - ID souboru v databázi
nickname: String - nové uživatelské jméno
Návratová hodnota:
Promise<null|String> - funkce nic nevrací pokud byl soubor úspěšně upraven, vrací ale chybovou hlášku, pokud nastane problém

ViewModel.getViews
Slouží k získání všech uložených pohledů v databázi.
Návratová hodnota:
Promise<Object[]> - seznam objektů reprezentující pohledy v databázi

ViewModel.getViewByID
Najde pohled v databázi podle ID a vrátí ho.
Parametry:
viewID: String - ID pohledu v databázi
Návratová hodnota:
Promise<Object> - objekt nalezeného pohledu z databáze

ViewModel.getAllViewsForFile
Najde všechny pohledy, které patří k danému souboru a vrátí je.
Parametry:
fileID: String - ID souboru v databázi
Návratová hodnota:
Promise<Object[]> - seznam objektů pohledů z databáze, které patří k danému souboru

ViewModel.addView
Přidá nový pohled do databáze.
Parametry:

title: String - uživatelský název pohledu
fileID: String - ID souboru v databázi
plotSettings: Object[] - Seznam objektů odpovídající strukturou strukturou plotSetting z 4.4

Návratová hodnota:

Promise<Object[]> - při úspěšném vytvoření vrátí ID nového objektu z databáze, při neúspěchu vrátí chybu

ViewModel.deleteView
Smaže pohled podle ID z databáze.
Parametry:

viewID: String - ID pohledu v databázi

Návratová hodnota:

Promise<null|String> - funkce nic nevrací pokud byl pohled úspěšně smazán, vrací ale chybovou hlášku, pokud nastane problém

ViewModel.editView
Změní údaje o pohledu v databázi.
Parametry:

viewID: String - ID pohledu v databázi
changes: Object - slovník obsahující jako klíče hodnoty, které se mají změnit a jejich hodnoty jsou poté hodnoty, které chceme uložit do databáze

Návratová hodnota:

Promise<null|String> - funkce nic nevrací pokud byl pohled úspěšně upraven, vrací ale chybovou hlášku, pokud nastane problém

ZoomModel.addZoom
Slouží k uložení nového úseku do databáze.
Parametry:

title: String - uživatelský název úseku grafu
sequence: [Number[]] - sekvence úseků z frontendové aplikace
viewID: String - ID pohledu v databázi, ke kterému se úsek vztahuje

Návratová hodnota:

Promise<Object> - při úspěšném uložení do databáze vrací celý nový záznam z databáze, pokud ale nastane chyba nevrací nic

ZoomModel.deleteZoom
Odstraní úsek z databáze
Parametry:
zoomID: String - ID úseku grafu v databázi
Návratová hodnota:
Promise<String> - funkce nic nevrací pokud byl pohled úspěšně smazán, vrací ale chybovou hlášku, pokud nastane problém

ZoomModel.getZoomsForView
Najde všechny úseky, které patří k danému pohledu a vrátí je.
Parametry:
viewID: String - ID pohledu v databázi
Návratová hodnota:
Promise<Object[]> - seznam úseků z databáze, které patří k danému pohledu

ZoomModel.updateZoom
Změní údaje o úseku v databázi.
Parametry:
zoomID: String - ID úseku grafu v databázi
changes: Object - slovník obsahující jako klíče hodnoty, které se mají změnit a jejich hodnoty jsou poté hodnoty, které chceme uložit do databáze
Návratová hodnota:
Promise<null|String> - funkce nic nevrací pokud byl úsek úspěšně upraven, vrací ale chybovou hlášku, pokud nastane problém

5.2 Tvorba frontend aplikace

Struktura výsledné aplikace

- /css – složka obsahující kaskádové styly
- /dist – výstup z webpacku
  - bundle.css – balík všech kaskádových stylů
  - bundle.js – balík všech javascriptových kódů
- /js – složka s javascriptovými soubory
  - /components – složka s všemi javascriptovými komponentami
    - /classes – složka pro datové objekty
      - Zoom.js – objekt reprezentující úsek grafu
    - /Managers – složka s manažery(modely)
      - ImageManager.js – manažer pro práci s obrázky
      - PlotManager.js – manažer pro práci s grafy
      - ZoomManager.js – manažer pro práci s úseky grafu
    - /Settings – složka obsahující komponenty nastavovacího procesu
      - /Pages – složka obsahující jednotlivé stránky modálního okna
      - DataUploadPage.js – stránka pro nahrávání souborů
      - FileSelectPage.js – stránka pro nahrávání souborů
      - IModalPage.js – rozhraní pro stránky modálního okna
      - ViewMakerPage.js – stránka pro tvorbu pohledu
      - ViewSelectPage.js – stránka pro výběr hotových pohledů
      - Modal.js – abstraktní třída modálního okna
      - SettingModal.js – modální okno nastavovacího procesu
    - LeftBar.js – komponenta levého panelu
    - Plot.js – komponenta grafu
    - PlotToolsTile.js – komponenta panelu pro práci s grafem
    - RightBar.js – komponenta pravého panelu
    - ZoomsTile.js – komponenta s úseky grafu
    - ZoomToolsTile.js – komponenta panelu pro práci s úseky
  - index.js – vstupní bod aplikace
- /node_modules – pomocné knihovny z npm
- index.html – HTML dokument fronend aplikace
- package.json – soubor s výčtem všech npm modulů a knihoven
- webpack.config.js – soubor s konfigurací webpacku

5.2.1 Index.js

Soubor index.js představuje hlavní bod pro Webpack a tudíž z tohoto bodu vedou reference do všech ostatních skriptů. I když aplikace nevyužívá žádný framework, který by definoval strukturu aplikace, tak některé principy byly z dnešních frameworků převzaty.

Aplikace přináší svůj systém komponent, tedy vizuálních částí stránky, které lze znovu použít, jsou napsané v javascriptu a starají o výsledný vzhled elementů. Komponenta je javascriptový objekt, který přijímá v konstruktoru selektor předvytvořeného elementu, který bude komponenta spravovat a následně slovník *options*, ve kterém mohou být dodatečné informace.

Komponenty, které mají být vytvořeny hned při načtení aplikace jsou definované právě v souboru index.js v seznamu *components*. Jak lze vidět v kódu 7, komponenty jsou zde zapsané v objektovém tvaru s atributy *class*, který určuje, jaká třída komponentu definuje, a již zmíněnými atributy *selector* a *options*.

```
const components = [
    {
        class:      Modal,
        selector:   '#modal-root',
        options: {
            'DOM_ID':       'settingsModal',
            'zoomManager':  zoomManager
        }
    },
...
```

Kód 7: Použití komponent v souboru index.js

Kód 8 ukazuje, jak je přes komponenty v seznamu iterováno. Pro každou komponentu se hledají elementy odpovídající selektoru, a pokud jsou nalezeny, tak se vytvoří příslušné třídy, které komponenty spravují. Další komponenty lze pak vytvářet i za běhu programu a není tak omezeno použití komponent jen na tento soubor.

```
components.forEach(component => {
    if(document.querySelector(component.selector) !== null){
        document.querySelectorAll(component.selector).forEach(
            element => new component.class(element, component.selector,
                                           component.options)
        )
    }
});
```

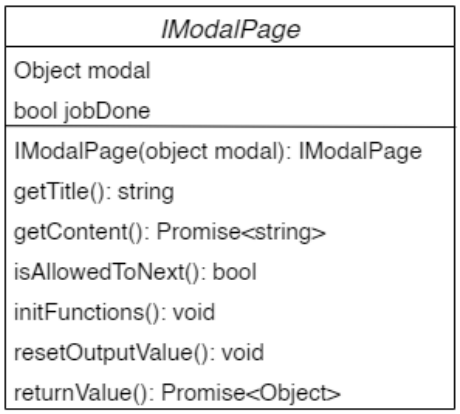
Kód 8: Použití komponent v souboru index.js

5.2.2 Nastavovací proces

Nastavovacím procesem se rozumí získání všech potřebných informací od uživatele k vytvoření grafů. Diagram toho procesu byl již popsán na obrázku 4.2 a chování výsledné aplikace z něj plně vychází.

O průběh nastavovacího procesu se stará komponenta SettingsModal.js. Komponenta představuje modální okno, které se skládá z různých stránek odpovídající právě stavům z diagramu nastavovacího procesu. Komponenta má v sobě definovaný seznam komponent stránek *content*, které se starají o interakci s uživatelem a získáváním informací od něj. Stránky se v modálním okně zobrazují v pořadí, v jakém se nachází v seznamu. Je proto důležité dbát na jejich pořadí pro správnou funkčnost.

Komponenty stránek vychází z třídy *IModalPage*, jejíž diagram je na obrázku 5.1. Třída definuje dva povinné atributy: *modal*, který přestavuje referenci na objekt modálního okna, pod který stránka spadá, a dále logickou hodnotu *jobDone*, která indikuje modálnímu oknu, jestli uživatel provedl na stránce všechny potřebné akce k tomu, aby bylo možné přejít na další stránku.



Obrázek 5.1: Třída IModalPage

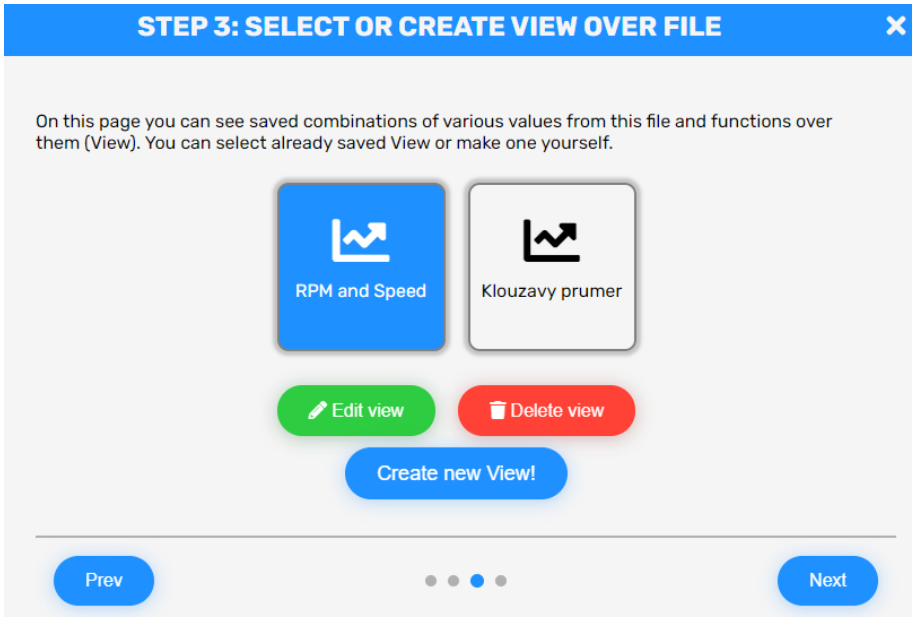
V konstruktoru stránky se předává odkaz na modální okno. Funkce *getTitle* slouží k tomu, aby modální okno vypsalo správný název stránky do nadpisu. *GetContent*

slouží obdobně k tomu, aby se do obsahu modálního okna vykreslil obsah stránky. Tato funkce tedy generuje veškeré texty, formulářové prvky a další elementy, které stránka využívá. Funkce *isAllowedToNext* slouží prakticky jako getter pro hodnotu *jobDone*, nicméně různé stránky zde mají i logiku, která kontroluje, zda je opravdu vše splněno pro přechod na následující stránku.

Funkci *initFunctions* volá modální okno po zobrazení obsahu a slouží k přiřazení event handlerů pro elementy stránky a různé změny stránky podle předchozích stavů stránek atd. Pokud se uživatel vrací v modálním okně na předchozí stránky, je volána funkce *resetOutputValue*, kde stránka může a nemusí reagovat na to, že se na stránku přistoupilo z druhého směru. Poslední funkce *returnValue* je volána při přechodu na další stránku a slouží k uložení informací z aktuální stránky do paměti modálního okna. Modální okno obsahuje slovník hodnot *data*, do kterého stránky ukládají své informace pro tvorbu grafu, a pro ostatní stránky, ve formátu klíč – hodnota, právě ve funkci *returnValue*.

Za nastavovacím procesem stojí aktuálně čtyři stránky, jejichž komponenty jsou: *FileSelectPage.js*, *DataUploadPage.js*, *ViewSelectPage.js* a *ViewMakerPage.js*.

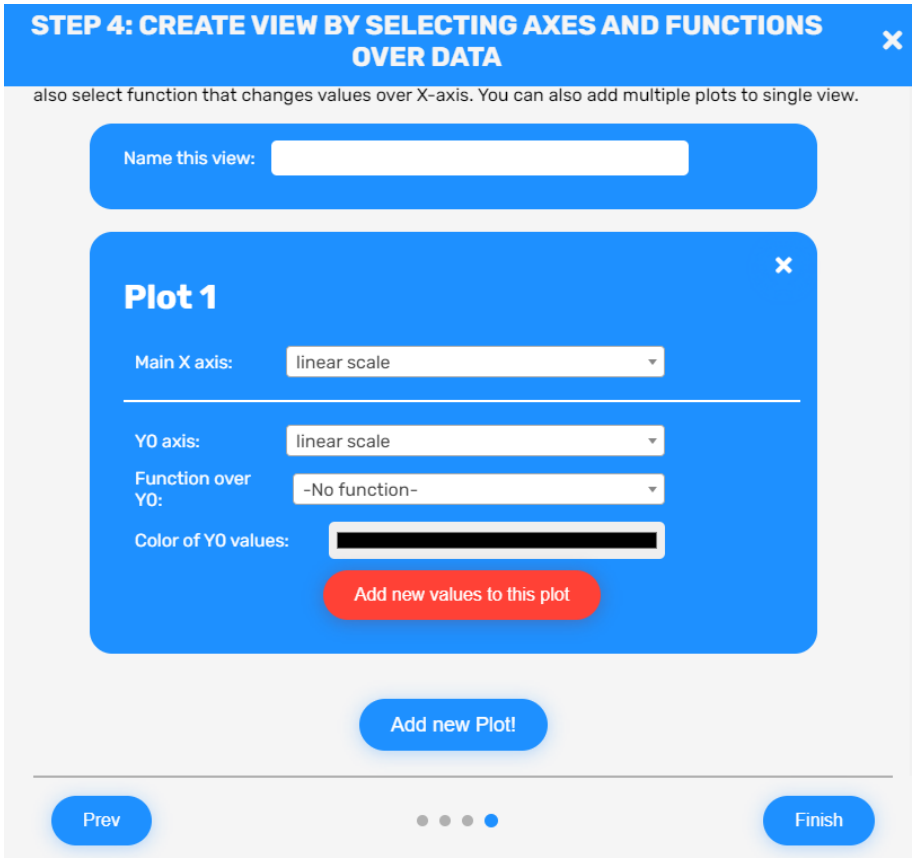
Při zobrazení modálního okna je jako první zobrazena stránka *FileSelectPage*. Ta slouží k zobrazení již nahraných souborů na serveru. Podobně jako lze vidět na obrázku 5.2, uživateli jsou zobrazeny nahrané soubory v jednotlivých buňkách uprostřed stránky. U souboru lze vidět jeho uživatelské pojmenování a datum nahrání na server. Uživatel kliknutím vybere soubor, se kterým chce pracovat a je přenesen na stránku *ViewSelectPage*. Stránka *DataUploadPage* je přeskočena, protože se stará o nahrávání nového souboru, což v tomto scénáři nedává smysl. Pokud by ale uživatel chtěl pracovat s novým souborem, klikne na tlačítko *Upload new file*, a bude přenesen na stránku *DataUploadPage*.



Obrázek 5.2: Obrázek okna pro výběr nahraných souborů

Stránka *DataUploadPage* dává uživateli možnost nahrát vlastní soubor na server. To se provede buď kliknutím na označený element, a následným vybráním souboru v prohlížeči nebo přetažením souboru do toho elementu. Aplikace odešle AJAXový požadavek na server, ve kterém se nachází nahraný soubor, a server soubor zpracuje. O poslání se stará funkce *parseFiles* definovaná v *DataUploadPage.js*. Pokud frontendová i backendová aplikace označí soubor za validní, uživateli se zobrazí pole, ve kterém může soubor pojmenovat. Po pojmenování se pošle druhý požadavek na server, který pojmenuje soubor. Je-li soubor nahraný a pojmenovaný, uživatel může přejít na další stránku, *ViewSelectPage*.

Na stránce *ViewSelectPage*, obrázek 5.2, jsou uživateli zobrazeny všechny vytvořené pohledy spojené s vybraným souborem. Aplikace si pamatuje vybraný soubor díky hodnotě *file* uložené v proměnné SettingsModal *data*. Při zobrazení stránky se tak pošle dotaz na server s hodnotou ID souboru a server vrátí uložené pohledy. Uživatel si jeden ze zobrazených pohledů vybere a pokud ho nechce nijak upravovat, tak po kliknutí na tlačítko *Next* je nastavovací proces u konce. Pokud uživatel chce pracovat s novým pohledem klikne na tlačítko *Create new View*, a je přenesen na poslední stránku nastavovacího procesu, *ViewMakerPage*.



Obrázek 5.3: Obrázek okna pro výběr nahraných souborů

Poslední stránka *ViewMakerPage* slouží k vytvoření nového pohledu nad daty ze souboru. Veškerá konfigurace výsledných grafů je dána právě pohledem. Uživatel

v první řadě musí pohled pojmenovat. Poté musí nastavit výsledné grafy. U každého grafu je potřeba nejdříve zvolit hodnoty pro osu X, kterou musí sdílet všechny řady v grafu. Tím je zajištěna komptabilita dat a jejich správné vykreslení do grafu. Hodnota osy X buď vychází z dat v souboru, popřípadě si uživatel může zvolit jako hodnotu X přednastavené měřítko. Aktuálně jsou k dispozici měřítka dvě: lineární a logaritmické.

Po zvolení osy X je potřeba zvolit hodnoty výsledné řady. Řad může být v grafu více, pokud splňují, že mají stejný rozměr. Přidání další řady se provede tlačítkem *Add new values to this plot*. Jednotlivým řadám pro odlišení lze vybrat rozdílné barvy kliknutím na formulářový prvek pro volbu barvy.

Pod jeden pohled může spadat i vícero grafů. Přidání nového grafu do pohledu lze udělat kliknutím na tlačítko *Add new Plot*. Tím se objeví stejné okno jako pro nastavení grafu 1 označeného na obrázku 5.3 jako *Plot 1*. Po nastavení grafů se celý nastavovací proces ukončí tlačítkem *Finish* v zápatí okna.

5.2.3 Zobrazování grafů uživateli

Po dokončení nastavovacího procesu má *SettingsModal* všechna potřebná data k vytvoření grafu v proměnné data. Po kliknutí na tlačítko *Finish* se spustí metoda *onFinishHandler*, která pošle na server dotaz o hodnoty pro vytvoření grafů. Konkrétně pošle dotaz na koncový bod /valuesFromView. API zpět vrátí data, a *SettingsModal* vyvolá globální událost *setupFinished*, které se předají tato obdržená data. Tím činnost *SettingsModal* končí a celé modální okno se skryje.

Na událost *setupFinished* zareaguje *PlotManager.js*. Tato třída se stará právě o vytváření komponent grafů. Z obdržených hodnot si zjistí, kolik je v aktuálním pohledu grafů a vytvoří prázdné svg elementy pro každý z nich. K nim podobně jako to dělá index.js vytvoří komponenty *Plot*, definované v /js/components/Plot.js a předá každé z nich hodnoty, které se jich týkají. Každý graf tak má jen pouze svá data.

Komponenta *Plot* po vytvoření zavolá funkci *parseData*, která převede data do konkrétního formátu pro každou časovou řadu, kterou graf obsahuje. Tato funkce je zobrazena v kódu 9. Výsledné hodnoty jsou uloženy v proměnné *data*, která představuje seznam, který má tolik prvků, kolik průběhů v grafu bude. Každý prvek se skládá z hodnoty *points*, která je pro grafy klíčová. Představuje totiž seznam dvojic hodnot x a y, pro každý jednotlivý bod průběhu. Hodnoty pro správné proložení křivkou musí být seřazeny podle času, případně podle hodnoty x. Dále se ukládá barva průběhu a ID představující číselnou hodnotu, která určuje, o kolikátý průběh grafu se jedná, dále se ukládjí ještě textové informace o názvu os a použité funkci nad daty.

```
parseData(data){
    this.data = [];
    return new Promise((resolve => {
        data.forEach((course, i) => {
            this.data.push({
                points: course.values,
                color: course.color,
                courseID: i,
                xColumn: course.xColumn,
                yColumn: course.yColumn,
                func: course.func
            })
        });
    }));
}
```

Kód 9: Funkce parseData

Po zpracování dat a připravení proměnných se volá funkce *init*, která nastaví svg elementu velikost. Je zde řešeno i odsazení grafu od okrajů elementu, kvůli prostoru pro popisy os. Dále se zde vytváří elementy, které budou ukazovat aktuální hodnotu grafu po najetí kurzoru nad bod grafu. Registrují se zde i eventlistenery, které reagují právě na pohyb myši nad grafem.

Kvůli možnosti přibližování a zobrazování úseku grafu se zde také vytváří takzvaná clipPath, což je speciální element, který určuje, co půjde na výsledném grafu vidět a co ne. Všechny hodnoty mimo tuto clipPath nebudou zobrazeny.

Je zde i vytvářena komponenta knihovny D3 s názvem *brushX*. Tato komponenta poskytuje rozhraní pro vybírání úseků grafu. Komponenta umí reagovat na dokončení výběru, a tak po dokončení výběru vyvolána nová globální událost *plotSelectionChanged*, na kterou reagují všechny grafy v aplikaci. Všechny grafy, tak vždy zobrazují tu samou část. Výběry jsou ale proporcionální a není tak důležité, že grafy mezi sebou nesdílí rozměry.

Knihovna D3 také ulehčuje samotnou tvorbu grafů, tím že poskytuje objekty a funkce, které přepočítávají data a zobrazují je. Nejdříve je potřeba stanovit měřítka os. Měřítko slouží k přepočítávání souřadnic bodů grafu mezi jeho šířku v prohlížeči. Je potřeba zjistit maxima a minima obdržených dat z csv souboru, a přesně k tomu slouží funkce *findBoundaries*. Po zjištění minim a maxim pro obě osy, je potřeba vytvořit měřítka. Každá osa má vlastní měřítko a kód pro vytvoření měřítka osy X je zobrazen v kódu 10.

```
this.xScale = d3.scaleLinear()
    .domain([boundaries.minX, boundaries.maxX])
    .range([0, this.width])
    .nice(50);
```

Kód 10: Tvorba měřítka pro osu X

Měřítku se předají nalezené rozmezí hodnot, a velikost, na kterou se mají hodnoty přepočítávat. V tomto případě je potřeba přepočítávat data mezi šířku elementu grafu a hodnotu nula.

Dále knihovna D3 poskytuje funkce k vytvoření os. Jak lze vidět v kódu 11, nejdříve je potřeba přidat do svg elementu nový element, ve kterém se budou osy nacházet, a následně se zavolá funkce po tvorbu osy, které se předá jako parametr vytvořené měřítko.

```
this.xAxis = this.svg
    .append("g")
    .attr('id', 'xAxis')
    .attr("transform", "translate(0," + this.height + ")")
    .attr("class", "axis")
    .call(d3.axisBottom(this.xScale));
```

Kód 11: Tvorba osy X

Po vytvoření os se vytváří ještě mřížka, která ulehčuje čtení hodnot z grafu. K tomu se využívá stejná funkce jako k tvorbě os, jen se za příkaz *axisBottom* přidají zřetězené funkce *ticks*, *tickSize* a *tickFormat*, které nastavují vzhled mřížky.

Takto je graf připravený, jen ještě neobsahuje žádné zobrazené řady. O to se stará kód funkce *drawPlot*. K zobrazení průběhů proložených křivkou je potřeba vytvořit další objekt knihovny D3, který se nazývá *line*. Jak lze vidět na kódu 12, jediné, co tato funkce potřebuje, je vědět, jak má zacházet s daty, které ji jsou poskytnuta a o jaký typ proložení se má jednat. V tomto případě stačí pouze definovat, že data, která obdrží má přepočítat podle vytvořených měřítek. Jako typ proložení je zvoleno *curveNatural*, které se nesnaží proložit body žádnou funkcí, ani rovnými čarami, ale co nejplynulejší křivkou.

```
this.lineFunction = d3.line()
  .x((data) => {
      return this.xScale(data.x);
  })
  .y((data) => {
      return this.yScale(data.y);
  })
  .curve(d3.curveNatural);
```

Kód 12: Tvorba objektu prokládající body křivkou

Když je vytvořena *lineFunction*, tak lze již vykreslit samotné grafy. V kódu 13 lze vidět, že s takto připravenými D3 objekty, stačí jen procházet data podle toho, k jakému průběhu patří, vytvořit jim element skupiny do hlavního svg elementu a následně vytvořit svg cestu, jejíž hodnoty sama vypočítá vytvořená *lineFunction* pro aktuální data. Pro aktuální průběh se ještě nastaví barva křivky a průběh je vykreslen.

```
this.data.forEach((course) => {
    // Append new element to SVG and store it in dictionary.
    this.lineGraphs[course.courseID] = this.svg.append("g")
      .attr("clip-path", "url(#"+this.clipId+")");
    this.lineGraphs[course.courseID]
      .append('path')
      .attr('id', 'lineGraph-path')
      .attr('class', 'curve')
      .attr('d', this.lineFunction(course.points))
      .attr('style', `stroke:${course.color};`);

});
```

Kód 13: Vykreslení křivek průběhů do grafu

5.2.4 Mechanismus práce s úseky grafů

Úseky grafů podle toho, jaké rozhraní poskytuje knihovna D3 funguje následovně. Při kliknutí a držení tlačítka myši se aktivuje komponenta *xBrush*, která označuje úsek grafu. Po puštění tlačítka myši funkce *end* komponenty *xBrush* vrátí počáteční a koncový pixel úseku grafu. Tyto pixely se berou z pozice uvnitř svg elementu. Podle toho lze změnit měřítko, a vše přepočítat, a je zobrazen jen úsek. Problém nastává při různých šířkách grafu, například změnou velikosti okna nebo při načtení úseku z databáze a použití na jiném zařízení s jiným rozlišením.

V této aplikaci jsou tedy úseky přepočítávány podle šířky elementu. V kódu 14 lze vidět, jak probíhá reakce na dokončení výběru u komponenty *xBrush*. Výsledný výběr se dělí šířkou elementu, od kterého je odečteno odsazení zleva. Výsledná hodnota je tak mezi hodnotami 0 a 1. 0 značí pozici na začátku grafu a 1 značí pozici na konci grafu.

Tento výběr je uložen do localStorage prohlížeče, aby k ní měly přístup všechny grafy a i jiné komponenty. Dále je vyvolána událost *plotSelectionChanged*, na kterou reaguje každý graf změnou zobrazeného úseku.

```
.on("end", (e) => {
    if(typeof(e.sourceEvent) !== 'undefined') {
        const event = new Event('plotSelectionChanged')
        event.from = 'mouse';
        const selection = e.selection !== null
          ? [
            e.selection[0]/(this.width-this.margin.left),
            e.selection[1]/(this.width-this.margin.left)
          ]
          : null;
        // Save selection as percentage of pixels selected from plot.
        localStorage.setItem('selection', JSON.stringify(selection));
        document.dispatchEvent(event);
    }
});
```

Kód 14: Reakce na výběr grafu

Když se poté reaguje na výše zmíněnou událost, je potřeba zpět přenásobit hodnoty úseku aktuální šířkou grafu, aby byla získána opět hodnota pixelů, mezi kterými byl úsek vybrán. Po přepočítání je volána funkce třídy *Plot* se jménem *updateChart*. Ta zjistí, jestli nemá výběr hodnotu *null*, a pokud má, tak nastaví úsek na maximum a minimum osy X a zobrazí tak celý graf.

Při zobrazování úseku je potřeba nejdříve podle měřítka zjistit body pro hodnoty pixelů a následně nastavit měřítko, tak aby na zadanou šířku grafu zobrazovala jen ty hodnoty, které jsou ve výběru. Jak lze vidět v kódu 15, tak toho lze dosáhnout díky D3 jediným příkazem.

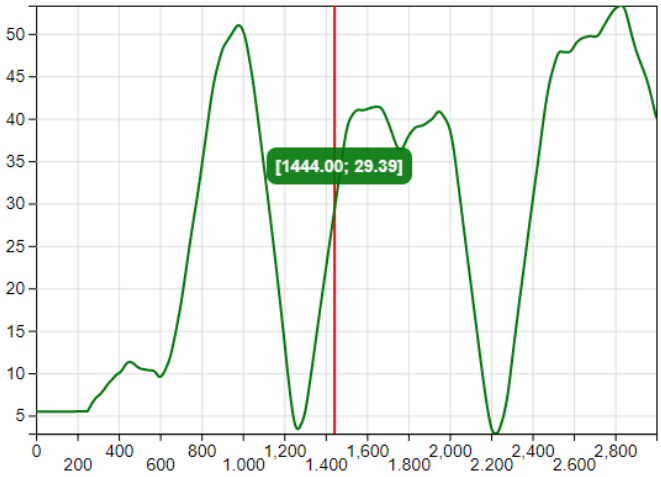
```
ref.xScale.domain([
    ref.xScale.invert(extent[0]),
    ref.xScale.invert(extent[1])
]);
```

Kód 15: Reakce na výběr grafu

Ostatní body, které jsou mimo toto měřítko jsou schované v dříve definované clipPath a nelze je tak vidět. Následně je potřeba přepočítat podle nového měřítka hodnoty osy X a změnit mřížku. Poté je potřeba ještě překreslit křivku prolínající body. Vše se provede stejně jako v definici, jen se použije nové měřítko.

5.2.5 Práce s grafy

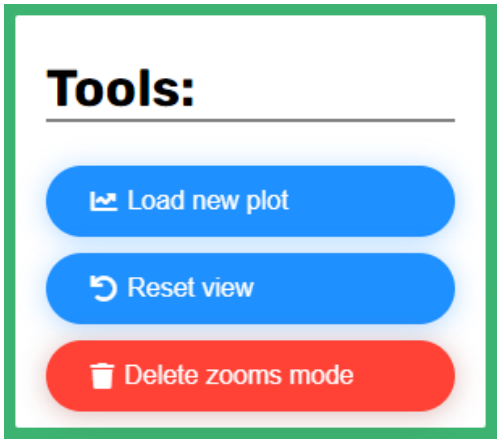
Nyní když jsou grafy nastavené a vykreslené, tak se s nimi může začít pracovat. Jak vypadá celé uživatelské rozhraní je možné vidět v příloze Příloha 1: Uživatelské rozhraní aplikace. Příklad vykresleného grafu je možné vidět na obrázku 5.4. Uživateli se zobrazuje na všech grafech svislá červená přímka, která znázorňuje polohu jeho kurzoru na grafu. Přímka se pohybuje v reálném čase společně s kurzorem. V místě, kde přímka protíná křivku průběhu se zobrazuje obdélník s aktuální hodnotou. Tato hodnota se počítá z pozice myši dosazených do měřítek os X a Y, podobně jako když se počítala hodnota výseku v kapitole 5.2.3. Když uživatel táhne kurzorem myši se stisknutým levým tlačítkem, provede se výběr úseku a graf se přiblíží jen na místo tohoto výseku.



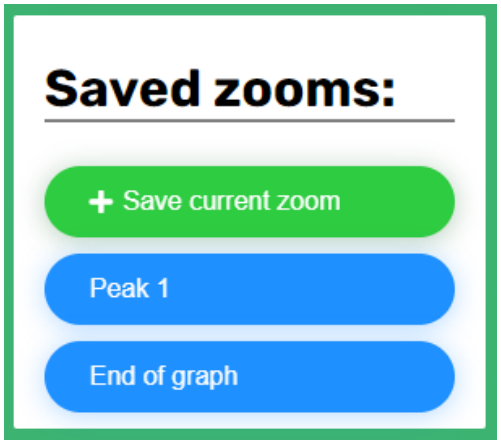
Obrázek 5.4: Obrázek vykresleného grafu

Uživateli jsou k dispozici v pravém sloupci vedle grafu nástroje, které může využít. Oba tyto panely nástrojů jsou definovány v komponentě RightBar.js. Ta při svojí inicializaci vytvoří další dvě komponenty a to: *ZoomToolsTile* na obrázku 5.5 a *ZoomTile* na obrázku 5.6.

ZoomToolsTile se skládá ze tří tlačítek. První, s nápisem Load new plot slouží k zobrazení *SettingsModal*, díky kterému lze načíst nové grafy. Druhé, s nápisem Reset view, slouží k znovu zobrazení celého grafu. *ZoomToolsTile* k tomu využívá metody *ZoomManageru*, který bude popsán dále. Poslední tlačítko s názvem *Delete zooms mode*, slouží k mazání uložených úseků grafu. Tlačítka v *ZoomTile* zčervenají a po kliknutí na ně, je možné je smazat. Po opětovném kliknutí na toto tlačítko se mazající mód vypne a tlačítka fungují jako původně.



Obrázek 5.5: Panel s nástroji



Obrázek 5.6: Panel s úseky grafu

Úloha *ZoomTile* je jednoduchá. Poskytuje možnost uložit aktuální úsek grafu po kliknutí na tlačítko *Save current zoom*. Dále zobrazuje všechny úseky spojené s aktuálně načteným pohledem. Úseky jsou reprezentovány tlačítky, na které když se klikne, tak se z grafu zobrazí jen uložená část. *ZoomTile* k tomuto také využívá *ZoomManager*. Pokud uživatel klikne rychle dvakrát po sobě na tlačítko úseku, tak může úsek přejmenovat.

ZoomManager je třída, která poskytuje funkce pro práci s úseky grafu. Třída pracuje s úseky tak, že všechny potřebné informace ukládá do localStorage. Když dojde k dokončení nastavovacího procesu a server vrátí úseky, které patří k danému pohledu, tak jsou tyto úseky uloženy do localStorage. Aplikace tak může s úseky grafu pracovat, aniž by pořád musela získávat z API hodnoty daných úseků. V kódu 16 je vidět zpracování globální události *plotSelectionChanged*, která nastane, když uživatel vybere úsek grafu. Samotné grafy tak nejsou jediné komponenty, které reagují na tuto událost. *ZoomManager* si při události vždy načte aktuální hodnotu úseku. Pokud načte hodnotu *null*, znamená to, že uživatel není aktuálně v žádném úseku.

```
onZoomChange(e) {
  const selection = JSON.parse(localStorage.getItem('selection'));
  if(selection === null){
    localStorage.setItem('zoomPath', JSON.stringify([null]));
  } else {
    const zoomPath = JSON.parse(localStorage.getItem('zoomPath'));
    zoomPath.push(selection);
    localStorage.setItem('zoomPath', JSON.stringify(zoomPath));
  }
}
```

Kód 16: Ukládání aktuální hodnoty úseku

Manažer pracuje s úseky jako se sekvencí. Aktuální hodnotu úseku vždy přidá do pole, kde jsou předchozí úseky. Vzniká tak sekvence, kterou když graf postupně osekáváme graf, dostaneme se na libovolný úsek, ať už uživatel provedl libovolný počet přiblížení po sobě. Sekvenci třída vždy nakonec opět uloží do localStorage.

Když uživatel chce uložit úsek na server, tak se zavolá metoda *saveCurrentZoom*, která načte aktuální sekvenci z localStorage a pošle jí na server k uložení do databáze. Zpět dostane její ID a další informace, které uloží do dalšího místo v localStorage, kde jsou uloženy všechny úseky daného pohledu. ID a další informace vrátí zpět do *ZoomTile*, odkud byla metoda volána, a vytvoří se tlačítko nového úseku.

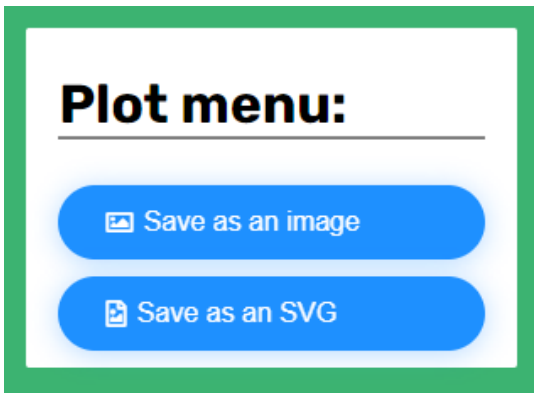
V localStorage je uložená i samotná hodnota úseku, se kterou pracují grafy, když se vyvolá událost *plotSelectionChanged*. Když se klikne na tlačítko úseku v *ZoomTile*, tak se provede jen postupné čtení z uložené sekvence, kdy se jednotlivé hodnoty s časovou prodlevou ukládají do toho místa v localStorage a vyvolává se událost *plotSelectionChanged*. Grafy na to reagují postupným zobrazením cíleného úseku. Ukázka kódu je kód 17. Jako první se jako úsek dává hodnota *null*. Tím se docílí toho, že se graf dostane do výchozí polohy, kde jde vidět celý, a tím je zaručeno, že výsledný úsek začíná vždy z pozice celého grafu, a ne jen z aktuálního úseku. Následně jsou volány už uložené hodnoty.

```
for(let i = 0; i < zoomSequence.length; i++) {
  setTimeout(() => {
    if(i === 0 && zoomSequence[i] === null)
      this.zoomManager.setSelection(null)
    else
      this.zoomManager.setSelection([
          zoomSequence[i][0],
          zoomSequence[i][1]]
      );
    localStorage.setItem('activeZoom', JSON.stringify(zoomID));
    this.zoomManager.fireChangeEvent('button');
  }, 300*i);
}
```

Kód 17: Kód rekonstrukce uživatelových výběrů částí grafu

Aplikace ještě obsahuje nástroje, které jsou k dispozici v levé straně aplikace. Tento panel je viditelný a aktivní pouze pokud uživatel předtím byl kurzorem nad nějakým z vytvořených grafů. Tento panel totiž obsahuje nástroje, které se týkají pouze jednoho grafu. Tento panel je vytvářen komponentou LeftBar.js, která stejně jako RightBar.js vytváří další komponenty při inicializaci. *LeftBar* ale vytváří pouze komponentu *PlotToolsTile*. Jak lze vidět na obrázku 5.7, tato komponenta se skládá ze dvou tlačítek, kdy obě mají úlohu exportovat aktuální graf do obrázku. První tlačítko exportuje graf do rastrového obrázku typu PNG a druhé tlačítko exportu-

je graf do vektorového formátu SVG. Obě tlačítka k tomu využívají funkce třídy *ImageManager*.



Obrázek 5.7: Panel nástrojů pro práci s grafem

Obě operace nejdříve při exportu využívají funkci *getSVGString*, která nejdřív funkcí *getCssStyles* získá styly spojené s grafem a jeho elementy. Princip funkce *getCssStyles*, je takový, že se uloží do seznamu všechny selektory, které se týkají alespoň některé z částí grafu. Poté se prochází všechny CSS soubory aplikace a hledají se definice stylů pro uložené selektory. Ty se ukládají do slovníku, kde je selektor jako klíč a styly jako hodnoty. Tento seznam se vrací zpět do hlavní funkce.

Následně se do SVG elementu grafu přidá nový style element, který se naplní nalezenými selektory se styly. Přes třídu *XMLSerializer* se celý SVG element i s potomky, tudíž i se style elementem, převedou na textový řetězec. Do řetězce jsou přidány náležitosti SVG souboru. A následný řetězec lze stáhnout jako SVG soubor.

Při extrahování do rastrové grafiky je potřeba udělat ještě nějaké operace navíc. Nejdříve se vytvoří nový element *Canvas*. Do toho elementu se vykreslí extrahovaný SVG řetězec. Hotový *Canvas* lze převést to struktury *Blob*, kterou lze následně stáhnout jako PNG obrázek.

6 Závěr

Výsledkem práce je webová aplikace, která řeší zobrazování rozsáhlých datových řad. Kompletní aplikace se skládá ze serverové aplikace, které komunikuje s databází a poskytuje vytvořené frontendové aplikaci potřebná data. Ze zkoumaných technologií na webovou vizualizaci dat byla vybrána knihovna D3.js, díky které aplikace umožňuje tvorbu mezi sebou komunikujících grafů, které se mohou skládat z libovolného množství průběhu z poskytnutých dat. V grafech je možné přibližovat na libovolně detailní úseky, které se ukládají v databázi, a tak je možné je používat i při ukončení aplikace. Úseky grafu lze exportovat do vektorové i rastrové grafiky. Nahraná data z csv souborů se ukládají do databáze a při jejich načítání je možné je pozměnit některou z vytvořených funkcí. Pro serverovou část byla zvolena technologie Node.js, která komunikuje s databází MongoDB.

Aplikace je plně zdokumentována, jak již v této práci, tak i v samotném kódu všech částí aplikace. Při návrhu a následné implementaci bylo dbáno na použití moderních objektových přístupů a strukturování kódu. Aplikace je tak připravena k dalšímu rozšiřování.

Aplikace byla testována na lokálním serveru a následně byla nahrána na zdarma poskytovaný webový hosting společností Heroku. Heroku se ale neosvědčilo jako dobrý hosting pro chod aplikace, kvůli jejich finančnímu modelu, který nepodporuje uchovávání dat, které nejsou součástí nahrané aplikace. Hosting tak průběžně maže nahrané csv soubory, a hlubší testování tak nebylo možné. Aplikace, pokud bude používána při práci na projektu Doprava 2020+, a poběží na univerzitním serveru, kde tento problém se soubory nenastane. Jednotlivé části nejsou pokryty testy, jak říká zadání, a to kvůli složitosti některých scénářů a závislosti na komunikaci frontendové části a backendové části. Nicméně na poskytnutých datech z projektu Doprava 2020+ je aplikace funkční na lokálním serveru a mimo problémy se soubory, je aplikace funkční i na serverech Heroku.

Aplikace je sice funkční a splňuje nároky ze zadání práce, i nároky konzultované s týmem Doprava 2020+, nicméně vizualizace dat je tak široký obor, že pro aplikaci je zde určitě velký prostor pro další vývoj. Mimo stoprocentního vyladění stávajících částí, jsou zde i body, které jsem měl v hlavě, ale nestihl implementovat.

Začnu od drobných detailů. U samostatných grafů jsem měl v plánu zobrazovat názvy sloupců, které byly použity při vykreslení, a následně je barevně přiřadit k vykresleným průběhům. Měl jsem také ambice, aby bylo možné měnit druh grafu, aby bylo možné vykreslovat hodnoty i jinak než jen do přímkového grafu. Aplikace také nabízí potenciál k použití mimo projekt Dopravy 2020+, to by ale aplikace musela zvládat rozlišování uživatelů, jejich autorizaci a autentifikaci. Poté by bylo

možné provozovat aplikaci veřejně, mimo univerzitní servery a mohla by tak posloužit jiným týmům, ale i jednotlivcům. Kdyby byla aplikace takto veřejná, uživatelé by určitě ocenili i možnost mezi sebou sdílet vytvořené pohledy, které představují nastavení grafů pro určitá data, popřípadě sdílet mezi sebou i nahrané soubory.

Použitá literatura

[1] *Program DOPRAVA 2020+: Vyhlášení 2. veřejné soutěže* [online]. [N.d.]. Dostupné také z: https://www.tacr.cz/program-doprava-2020-vyhlaseni-2-verejne-souteze/.

[2] *HTML 5* [online]. 2008. Dostupné také z: https://www.w3.org/TR/2008/WD-html5-20080122/.

[3] *Selectors Level 3* [online]. 2011. Dostupné také z: https://web.archive.org/web/20140603165900/http:/www.w3.org/TR/selectors/.

[4] *Standard ECMA-262 5.1 Edition* [online]. 2011. Dostupné také z: https://262.ecma-international.org/5.1/.

[5] *GitHub.com* [online]. [N.d.]. Dostupné také z: https://github.com/.

[6] KASHYAP, Neeraj. GitHub's Path to 128M Public Repositories. *Towards data science*. 2020. Dostupné také z: https://towardsdatascience.com/githubs-path-to-128m-public-repositories-f6f656ab56b1.

[7] BEUKE, Fabian. *GitHut 2.0* [online]. 2021. Dostupné také z: https://madnight.github.io/githut/.

[8] BOROVIKOV, Ruslan. 10 Best JavaScript Charting Libraries for Any Data Visualization Need. *DZone*. 2019. Dostupné také z: https://dzone.com/articles/10-best-javascript-charting-libraries-for-any-data.

[9] IVAYLO GERCHEV, Syed Fazle Rahman. 18+ JavaScript Libraries for Creating Beautiful Charts. *Sitepoint*. 2019. Dostupné také z: https://www.sitepoint.com/best-javascript-charting-libraries/.

[10] *AnyChart - JS charts* [online]. 2021. Dostupné také z: https://www.anychart.com/products/anychart/overview/.

[11] BOSTOCK, Mike. *D3 -Data-Driven Documents* [online]. 2021. Dostupné také z: https://d3js.org/.

[12] BOSTOCK, Mike. *D3.js Gallery* [online]. 2020. Dostupné také z: https://observablehq.com/@d3/gallery.

[13] *Charts.js*. 2020. Dostupné také z: https://www.chartjs.org/.

[14] *Plotly* [online]. 2021. Dostupné také z: https://plotly.com/.

[15] MAURI, Michele et al. RAWGraphs: A Visualisation Platform to Create Open Outputs. In: *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter*. Cagliari, Italy: ACM, 2017, 28:1–28:5. CHItaly '17. ISBN 978-1-4503--5237-6. Dostupné z DOI: 10.1145/3125571.3125585.

[16] *About Chart studio* [online]. 2021. Dostupné také z: https://plotly.com/chart-studio/.

[17] *Chart studio application* [online]. 2021. Dostupné také z: https://chart-studio.plotly.com/create/.

Seznam kódů

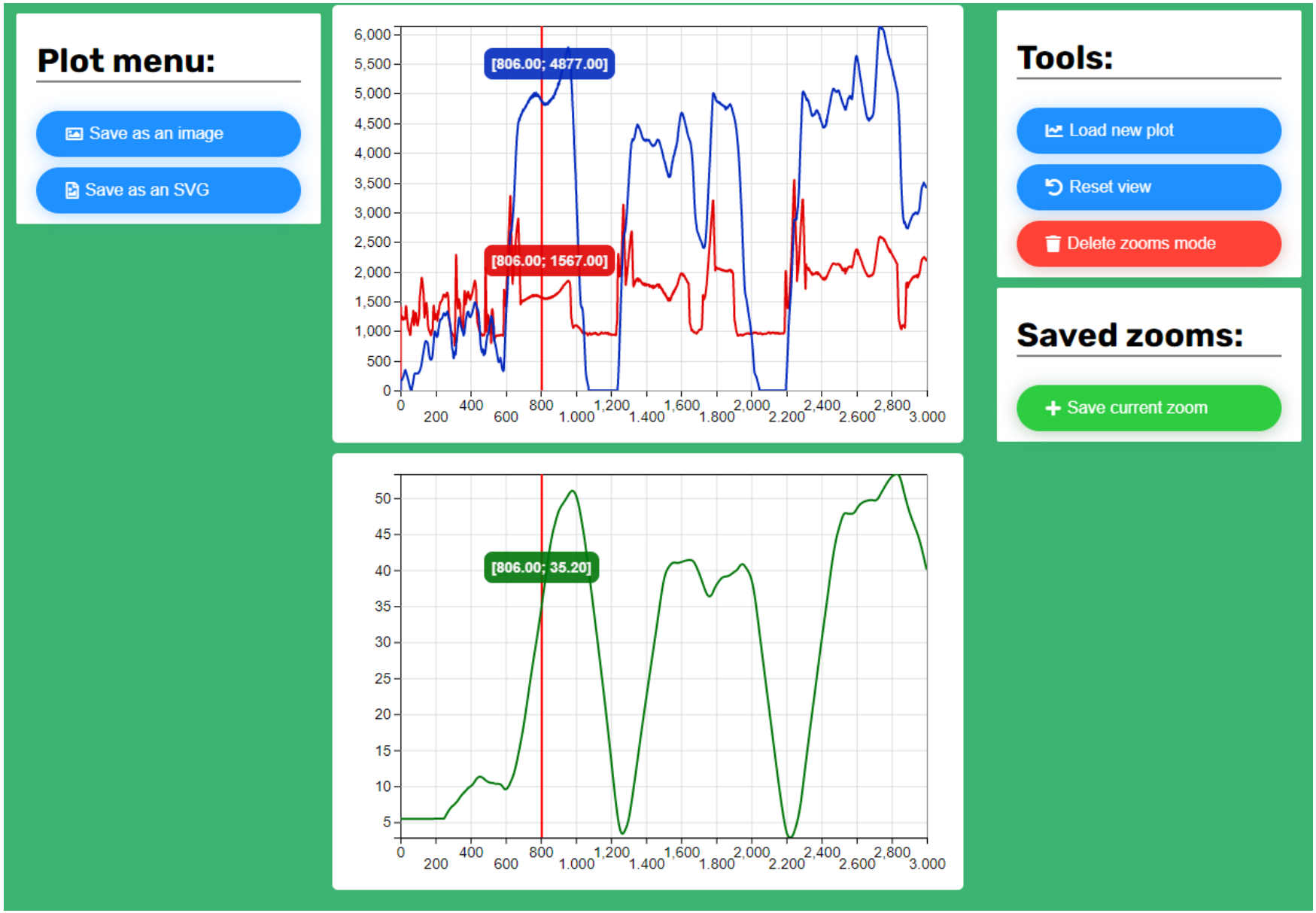
Seznam obrázků

Seznam tabulek

Přílohy

Příloha 1: Uživatelské rozhraní aplikace

(Příloha na druhé straně)


Plot menu:
Save as an image
Save as an SVG
[806.00; 4877.00]
[806.00; 1567.00]
[806.00; 35.20]
Tools:
Load new plot
Reset view
Delete zooms mode
Saved zooms:
Save current zoom